NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

SERVICE ORIENTED ARCHITECTURE FOR COAST GUARD COMMAND AND CONTROL

by

Russell E. Dash
Robert H. Creigh

March 2007

Thesis Advisor:	Rick Hayes-Roth
Second Reader:	Rex Buddenberg

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE: Service Oriented Architecture for Coast Guard Command and Control		**5. FUNDING NUMBERS**
6. AUTHOR(S) Russell Dash and Robert Creigh		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The Coast Guard's software architecture does not meet the organization's needs for information sharing or command and control. The Commandant of the Coast Guard recently mandated the implementation of a Service Oriented Architecture (SOA) to address this problem. This thesis describes a Service Oriented Architecture for Coast Guard Command and Control that integrates legacy applications and provides new capabilities. Traditional software architecture descriptions make it difficult to identify and understand the trade-offs between quality attributes that are inherent in the design. We clarify these critical issues by using multiple scenarios and use cases, in addition to diagrams and functionality requirements. Defining the architecture in this manner enables an auditor to determine the architecture's validity. The Coast Guard also needs a plan to implement this SOA. This thesis defines a process that will deliver value in the form of usable capabilities in an incremental manner. It recognizes the constantly changing nature of both the problem and the necessary solution, and evolves accordingly. It continually plans for, adapts to, and exploits predictable advances in technology to deliver more value. The iterative method we propose includes cyclical evaluation of the system requirements, architecture, and implementation to provide continuous improvement.

14. SUBJECT TERMS Service Oriented Architecture (SOA), Command and Control (C2), Coast Guard (USCG), C4ISR, Web Services, Extensible Markup Language (XML), Software Architecture Evaluation, Architecture Tradeoff Analysis Method (ATAM), Software Architecture Implementation.			**15. NUMBER OF PAGES** 151
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

SERVICE ORIENTED ARCHITECTURE FOR COAST GUARD COMMAND AND CONTROL

Russell E. Dash
Lieutenant Commander, United States Coast Guard
B.S., United States Coast Guard Academy, 1994

Robert H. Creigh
Lieutenant, United States Coast Guard
B.S., University of Phoenix, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 2007

Authors:	Russell E. Dash
	Robert H. Creigh
Approved by:	Rick Hayes-Roth Thesis Advisor
	Rex Buddenberg Second Reader
	Dan Boger Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Coast Guard's software architecture does not meet the organization's needs for information sharing or command and control. The Commandant of the Coast Guard recently mandated the implementation of a Service Oriented Architecture (SOA) to address this problem. This thesis describes a Service Oriented Architecture for Coast Guard Command and Control that integrates legacy applications and provides new capabilities. Traditional software architecture descriptions make it difficult to identify and understand the trade-offs between quality attributes that are inherent in the design. We clarify these critical issues by using multiple scenarios and use cases, in addition to diagrams and functionality requirements. Defining the architecture in this manner enables an auditor to determine the architecture's validity. The Coast Guard also needs a plan to implement this SOA. This thesis defines a process that will deliver value in the form of usable capabilities in an incremental manner. It recognizes the constantly changing nature of both the problem and the necessary solution, and evolves accordingly. It continually plans for, adapts to, and exploits predictable advances in technology to deliver more value. The iterative method we propose includes cyclical evaluation of the system requirements, architecture, and implementation to provide continuous improvement.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACRONYMS AND ABBREVIATIONS

ADO	Assistant Duty Officer
AIS	Automatic Identification System
AMVER	Automated Merchant Vessel Reporting
AOPS	Abstract of Operations
BPEL4WS	The Business Process Execution Language for Web Services
BPMN	Business Process Modeling Notation
CASP	Computer Aided Search Planning
C2	Command and Control
C2CEN	Command and Control Engineer Center
CC	Command Center
CCPM	The Coast Guard Command Center Program Manual
CDM	Conceptual Data Model
CDNU	Cockpit Display Navigational Unit
CDO	Command Duty Officer
CG	Coast Guard
CGC	Coast Guard Cutter
CGC2	Coast Guard Command and Control
CGDN	Coast Guard Data Network
CGMS	Coast Guard Message System
CMA	Comprehensive Maritime Awareness
CO	Commanding Officer
COI	Communities of Interest
COPORD	Common Operational Picture Operational Requirements Document
CTO	Chief Technical Officer
DHS	Department of Homeland Security
DoD	Department of Defense
DoDAF	DoD Architecture Framework
ELT	Emergency Locator Transmitter
eNOAD	Electronic Notice of Arrival/Departure
EPIRB	Emergency Position Indicating Radiobeacon
FTP	File Transfer Protocol
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICS	Incident Command System
IDeA	Incremental Development Approach
IP	Internet Protocol
IT	Information Technology
JAWS	Joint Automated Worksheet

JC3IEDM	Joint Consultation Command & Control Information Exchange Data Model
JCTD	Joint Capability Technology Demonstration
JRCC	Joint Rescue Coordination Center
LEDET	Law Enforcement Detachement
LEDO	Law Enforcement Duty Officer
LNM	Local Notice to Mariners
M/V	Motor Vessel
MIEM	Maritime Information Exchange Model
MISLE	Marine Information for Safety and Law Enforcement
MHS-OPS	Maritime Homeland Security Operational Planning System
MSRT	Maritime Security Response Team
MSST	Maritime Safety and Security Team
NCO	Network Centric Operations
NIEM	National Information Exchange Model
NMS	Network Management System
NPS	Naval Postgraduate School
OPAREA	Operational Area
OPCEN	Operations Center
OPTEMPO	Operational Tempo
OSC	Operations Systems Center
PAWSS	Ports and Waterways Safety System
PLB	Personal Locator Beacons
PKI	Public Key Infrastructure
QoS	Quality of Service
R21	Rescue 21
R21DF	Rescue 21 Direction Finder
SAML	Security Assertion Markup Language
SANS	Ship Arrival and Notification System
SAR	Search And Rescue
SARSAT	SAR Satellite
SDL	Service Development Loop
SEI	Software Engineering Institute
SITREP	Situation Report
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPS	Search Pattern Service
TCP	Transmit Control Protocol
TISCOM	Telecommunications and Information Systems Command
UDDI	Universal Description Discovery and Integration
USCGC	United States Coast Guard Cutter
USMCC	United States Mission Control Center
VBST	Vessel Boarding and Search Team

VHF	Very High Frequency
VMS	Vessel Monitoring System
VTS	Vessel Traffic Service
W3C	World Wide Web Consortium
WAN	Wide Area Network
WSDL	Web Service Description Language
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XO	Executive Officer
XSD	XML Schema Documents
XSLT	Extensible Stylesheet Language Transformation

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

We would like to thank Dr. Rick Hayes-Roth, Rex Buddenberg, Dr. Alex Bordetsky, and Glenn Cook for your dedication and devotion to our educational journey. Your support and direction made our experience at the Naval Postgraduate School worthwhile. We both feel fortunate to have been your students.

LCDR Rusty Dash

To Danielle, this wouldn't exist without you. Thanks for giving way more than you received.

To Bradley, Bryce, and Brendon, one day you might actually peruse this thesis thing and wonder why it stole so much time otherwise reserved for us. Thanks for being great boys and giving me time to get this thesis done.

To Brian Rideout and the various members of Group TLR, thanks for the adventures in class projects and academic excellence. Friends like you made my time here "*a awesome*" experience.

To Bob, my thesis partner, thanks for such a great thesis topic. It meant a lot to have another Coastie to go through NPS with. Oh, and one more thing, I just found another great article that we can use in chapter…

LT Bob Creigh

To my wife Jill, thank you for supporting me throughout my career and during this time at NPS. Your love and devotion have not gone unnoticed.

To Andy and Heather, for all of the time I spent on this instead of with you doing things that we love to do.

To Rusty, you provided me almost as much education during my time here as the faculty. Thank you for all your effort toward writing this thesis and for being a good and trusted friend.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The roles and missions of the United States Coast Guard have changed significantly from the vision of its founding father, Alexander Hamilton, who stated that “A few armed vessels, judiciously stationed at the entrances of our ports, might at a small expense be made useful sentinels of the laws.” (Hamilton 1787) Today’s Coast Guard is a dynamic, multi-mission maritime organization dedicated to protecting the lives, safety, and security of the American people. This service is a unique combination of military combatant, law enforcement authority, and humanitarian do-gooder that the government and American public have come to expect will always be “Semper Paratus.” As such, it has been assigned a diverse set of strategic goals and missions that require partnership and interoperability with many local, state, federal, and international agencies, as well as the maritime industry and foreign governments. The five strategic goals and twenty major missions of the United States Coast Guard are:

Maritime Safety – Eliminate deaths, injuries, and property damage associated with maritime transportation, fishing, and recreational boating. The specific missions are Search and Rescue (SAR), Marine Safety Program, Recreational Boating Safety, and the International Ice Patrol.

National Defense – Defend the nation as one of the five U.S. armed services. The specific missions include Defense Readiness, Homeland Security, Ports Waterways and Coastal Security, and Polar Icebreaking.

Maritime Security – Protect America's maritime borders from all intrusions by: (a) halting the flow of illegal drugs, aliens, and contraband into the United States through maritime routes; (b) preventing illegal fishing; and (c) suppressing violations of federal law in the maritime arena. The specific missions are Illegal Drug Interdiction, Migrant Interdiction, Living Marine Resource Protection, General Maritime Law Enforcement, Exclusive Economic Zone Enforcement, and Treaty Enforcement.

Maritime Mobility – Facilitate maritime commerce and eliminate interruptions and impediments to the efficient and economical movement of goods and people, while maximizing recreational access to and enjoyment of the water. The specific missions are Aids to Navigation, Icebreaking Operations, and Vessel Traffic/Waterways Management.

Protection of Natural Resources – Eliminate environmental damage and the degradation of natural resources associated with maritime transportation, fishing, and recreational boating. The specific missions include Marine Environmental Science, Foreign Vessel Inspections, and Marine Pollution Response and Enforcement. (“Missions”)

Coastguardsmen are policemen, sailors, warriors, humanitarians, regulators, stewards of the environment, diplomats, and guardians of the coast while performing those missions. (*America’s Maritime Guardian* 2) Each of those duties has unique requirements for the type, amount, and complexity of information that must be managed. This information diversity is plainly visible when one considers the list of activities and accomplishments during an “average Coast Guard day.”

Every day the U.S. Coast Guard:

- Conducts 82 search and rescue cases
- Saves 15 lives
- Assists 114 people in distress
- Protects $4.9 million in property
- Boards 202 vessels of law enforcement interest
- Interdicts 26 illegal migrants at sea
- Seizes $12.4 million worth of illegal drugs

- Conducts 23 waterfront facility safety or security inspections
- Enforces 129 security zones
- Monitors the transit of 2,557 commercial ships through U.S. ports
- Boards 122 large vessels for port safety checks
- Boards 4 “high interest” vessels
- Investigates 20 vessel casualties involving collisions, allisions and groundings
- Responds to 11 oil and hazardous chemical spills
- Conducts 317 vessel safety checks
- Teaches 63 boating safety courses
- Conducts 19 commercial fishing vessel safety exams
- Processes 280 mariner licenses and documents
- Services 140 aids to navigation

(“Average Day”)

With such a high volume of daily activity in so many different mission areas, the Coast Guard faces a daunting information and communication problem. It needs to efficiently process and effectively utilize large amounts of varied information that typically originates from unplanned events. Unfortunately the Coast Guard is burdened with an information technology (IT) infrastructure composed of standalone applications and communications networks that lack interoperability. The combination of heterogeneous missions, applications, and networks creates information sharing problems within the Coast Guard and with external entities that result in operational inefficiency and ineffectiveness. In addition the Coast Guard has become an integral part of the rapidly evolving, extended homeland security enterprise that spans multiple federal departments and reaches out to many state and local government agencies. This means the information sharing needs of the Coast Guard are ever growing and will be increasingly influenced by its partners, both within the federal government and beyond. The September 11th 2001 terrorist attacks and Hurricane Katrina highlighted weaknesses in our nation's intra- and inter-agency information sharing and "demonstrated the critical need for developing improved (distributed, shared and fault-tolerant) enterprise governance systems that are at once stand-alone and interoperable." (Bayne 14) In response to these challenges, the Coast Guard must develop a credible architecture and then adopt a flexible, rapid, and incremental implementation process.

B. SOFTWARE ARCHITECTURE

Enterprise level software architectures connect business goals and computer systems by describing the structures of the software elements, including the externally visible properties of the elements, and the relationships and interactions between them. "Externally visible" properties refers to those assumptions that other elements can make about the behavior of an element, such as its provided services and performance characteristics. The details of elements that have solely to do with internal implementation are by definition not architectural. The architecture provides the fundamental organization of the system and the principles that govern its design and evolution. ("Published Software Architecture Definitions")

Successful software architectures are designed to meet both functional and quality attribute requirements. The functional requirements define what the software components do, and these are typically written in brief scenarios called use cases. An example of a functional requirement is: *given the necessary six input parameters (Commence Search Point, Length, Width, Major Axis, Track Spacing, First Turn), calculate the waypoints for a parallel search pattern as defined by the National SAR Manual and output them in Extensible Markup Language (XML) format that complies with the Joint Consultation Command & Control Information Exchange Data Model (JC3IEDM) schema*. Quality attributes are the benchmarks that describe a system's intended behavior within the environment for which it was built. They provide the means for measuring the fitness and suitability of a product. Quality attribute requirements such as those for performance, security, modifiability, reliability, and usability have a significant influence on the software architecture of a system. ("Software Architecture Glossary")

1. Software Architecture for Command and Control

The Coast Guard does not have a viable enterprise level software architecture that meets its current needs, let alone its rapidly evolving future needs. It is burdened with a collection of "stovepipe systems" that were individually created to address specific functional needs, without consideration or design for current functionality and quality-attribute requirements. Each stovepipe embeds the semantics of the data and the processing logic (functions) within the system. This configuration prevents other programs from accessing either the data or functions, effectively trapping them within the stovepipe. Because it is extremely difficult to integrate stovepipes, new systems often repeat data and functions which produces two serious problems. The first problem is that data about the same object (vessel, report, etc.) often differs from one stovepipe to the next, which creates confusion and uncertainty for the users. The second problem is the limitation in the number of systems a human can simultaneously utilize. Relevant data and useful functionality may go unused because they are too difficult to access and not all users have access to every stovepipe. To solve this problem the Coast Guard must integrate the data and functionality from the stovepipes in an new architecture that supports its rapidly evolving needs.

This thesis will focus on the development of a software architecture for Coast Guard Command and Control. Coast Guard Publication 1 defines Command and Control as "the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission." This includes planning, directing, coordinating, and controlling forces and operations to accomplish the mission. (*America's*

Maritime Guardian 60) Many theories about command and control break the decision making process into sense-decide-act stages that form an iterative loop. Dr. Rick Hayes-Roth further refines this theory by defining efficient thought by intelligent beings (person, organization, system). The functions of efficient thinking divide into eight steps, each supported by a world model that represents the intelligent being's understanding of how things work. "The world model provides the knowledge that an intelligent being uses to interpret events, generate candidate plans for improving situations, and select the most attractive candidates for execution." (Hayes-Roth, *Hyper-beings* 58)

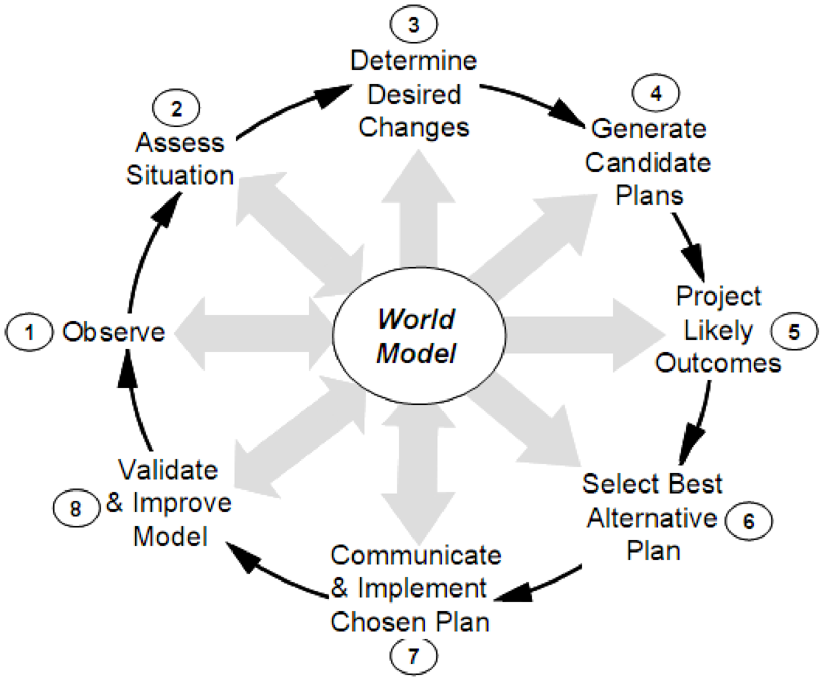

3 Determine Desired Changes
4 Generate Candidate Plans
Project Likely Outcomes 5
Select Best Alternative Plan 6
Communicate & Implement Chosen Plan 7
8 Validate & Improve Model
1 Observe
2 Assess Situation
World Model

Figure 1. Efficient Thought (From: Hayes-Roth *Hyper-beings* Fig 2.)

The eight steps of efficient thought are numbered in a typical sequence, though in most complex organizations all eight steps operate in parallel. "The intelligent being (1) observes what's happening in the environment, (2) assesses the situation for significant threats and opportunities, (3) determines what changes are desirable, (4) generates candidate plans for making those changes, (5) projects the likely outcomes of those plans, (6) selects the best plan, and (7) communicates that plan to key parties and implements it. Throughout, the intelligent being (8) validates and improves its model. The model supports all eight activities, although only steps 1, 2, 7 and 8 directly update and modify the model." (Hayes-Roth *Hyper-beings* 59) Software architecture for command and control needs to support the use of these eight steps.

2. Coast Guard Command and Control Architecture

The Coast Guard Command Center Program Manual (CCPM) describes a system model that depicts the fundamental components of command center performance as seven capabilities that produce three outputs. This system model relies upon the interaction between the capabilities of planning, execution, information collection, information processing, information sharing, awareness, and assessment to produce information management, situational awareness, and command and control. While not identical, many similarities exist between the Coast Guard's model and Dr. Hayes-Roth's efficient thought process described above.

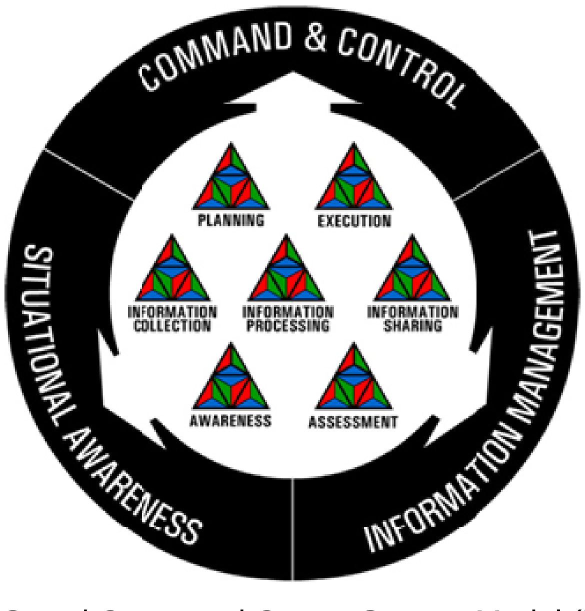


Figure 2. Coast Guard Command Center System Model (From: *Command Center Program Manual* Figure 1-2-1)

The authors of the CCPM use a triangle of triangles to represent each of the seven capabilities in the system model. Each “capability” triangle is composed of three smaller triangles; a blue one representing agents (human and software) to perform tasks, a green one representing infrastructure (computers, sensors, etc.), and a red one representing doctrine. The manual lists 35 different stovepipe software applications watchstanders can use to perform their duties. However, the system model does not describe how these elements function and interact to produce the required outputs. Absent this critical analysis and documentation, the current system model will never reliably produce the desired results.

Clearly the Coast Guard needs to find new and better ways of managing information and providing capabilities in response to quickly changing needs. It needs to design a component-based architecture that provides the necessary functionality with the required quality attributes. The Coast Guard will always have limited resources and its command and control requirements will continue to change over time. Therefore, any new architecture must facilitate integration with legacy systems in a way that reuses existing assets and allows flexible reconfiguration of both existing and new assets as needed. The architecture should also enable an evolution from the current state to required functionality that delivers value at each step along the way.

3. Service-Oriented Architecture (SOA)

The Commandant of the Coast Guard has mandated the implementation of a Service-Oriented Architecture to “better serve the needs of all our internal and external customers.” (Allen) The SOA methodology will supposedly enable the Coast Guard to reduce the expense of integration, increase asset reuse, and increase business (organizational) agility. SOA encapsulates the distinct functions contained in enterprise applications into loosely-coupled, interoperable, standards-based services that interact via a common communications protocol. “A service is an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes.” (Obrien 1) These services are combined and reused to meet the requirements of business processes and software users.

The Coast Guard does not have resources to simultaneously redevelop legacy system functionality and implement the new elements of SOA. The rich capabilities contained within legacy Command and Control (C2) applications can be reused in an SOA with a “wrapper.” This approach provides a viable economic option, because it avoids re-writing the existing software. Once service enabled, these legacy components can remain operationally intact within the current architecture and be made available as services at the same time. “SOAs are flexible because each service encapsulates the underlying platforms and technologies that support it. The services provided at the enterprise level are therefore agnostic to those specific platforms and technologies.” (Lau 11)

Unfortunately SOA does not provide the perfect solution to all the Coast Guard’s information sharing and application integration needs. SOA means different things to different people and the Coast Guard needs to have a clear understanding of the differing technologies, standards, and implementation methods. Because SOA holds so much promise, all the major software manufacturers and vendors are promoting their support with some directly involved in developing open standards. As a result, every major development platform now officially supports the creation of “service-oriented solutions.” (“The SOA Vision”) This competition between vendors with different standards must be approached with caution as it may actually make it more difficult to successfully develop a meaningful SOA to meet the Coast Guard’s needs. The next chapter will examine the standards and technologies used to implement SOAs with specific recommendations.

C. THESIS QUESTIONS

This thesis aims to provide sound, supported, informative, and valuable answers to Coast Guard IT decision-makers for the following two questions.

1. How Can the Coast Guard Implement a Service-Oriented Architecture for Command and Control?

Traditional descriptions of software architectures make it difficult to identify and understand the trade-offs between quality attributes that are inherent in the design. We will use multiple scenarios and use cases, in addition to diagrams and functionality requirements, to make these critical issues easier to understand. Defining the architecture in terms of the functionality and quality attribute levels of each component should allow an auditor to determine the architecture's validity. Our answer to this question does not create a complete architecture, however it does establish an effective starting point for the Coast Guard.

2. What is the Optimal Implementation Plan for this Coast Guard Command and Control (CGC2) SOA?

Almost all large scale software and IT system projects fail, so a "big bang" approach to create this SOA should be avoided. Because the entire SOA will not be created at the same time, the Coast Guard needs a process that will deliver value in the form of usable capabilities in an incremental and iterative manner. This sequence of capabilities should determine how the components and architecture evolve. Our proposed iterative method will include an evaluation of the system requirements, architecture, and implementation plan during each repetition of the cycle that guarantees continuous improvement. This plan will also incorporate industry best practices to anticipate and address predictable problems.

D. THESIS ORGANIZATION

Chapter I has defined the problem and introduced the basic approach for our solution. The remaining chapters of this thesis are organized as follows:

- Chapter II provides a synopsis of SOA components and standards.
- Chapter III describes an SOA for Coast Guard Command and Control.
- Chapter IV proposes an Implementation Plan for that architecture.

- Chapter V contains our conclusions and recommendations for future research*.*

THIS PAGE INTENTIONALLY LEFT BLANK

II. SOA BACKGROUND INFORMATION

A. SERVICES ORIENTED ARCHITECTURE

This chapter provides background information about SOA, Web services standards, and data models for readers new to the subject so they can grasp the material presented in the remaining chapters.

Service-Oriented Architecture is a software design methodology that uses loosely-coupled services to perform business functions or processes. These services communicate using well-defined standards across a network. Section C describes services and service-oriented design principles in detail. Services send XML-formatted messages that relay information structured in accordance with an accepted data model. Section B defines the basic XML terms and Section E discusses data models.

SOA proponents believe it can help businesses (and government agencies) respond more quickly and cost effectively to changing environmental conditions. "All major software manufacturers and vendors promote support for SOA – some even through direct involvement in the development of open standards. As a result, every major development platform now officially supports the creation of service-oriented solutions." ("The SOA Vision") While that statement sounds like a boon for businesses and government organizations considering an SOA, competing standards and vendors can actually make it more difficult to separate the marketing hype from the truly valuable technology to determine a path to success. We outline the core SOA standards in Section D. This chapter concludes with an example Web service in Section E.

B. XML

The Extensible Markup Language (XML) is an open standard for exchanging structured documents and data over the Internet. Authors and "…designers create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations." ("XML") A schema provides a framework for naming and storing different elements of information. XML Schema Documents (XSD) use XML to describe the schema for a certain kind of XML document or message. An XML message recipient can use the appropriate XSD to verify the message's data structure and format using a process called validation. Using XML to carry both the meta data and the data in the same message, composed using an agreed upon schema, begins to solve the data interoperability problem.

Because XML documents contain standard structure with the content, they can be easily converted to comply with another XML schema. XML Transformation documents, written in the Extensible Stylesheet Language Transformation (XSLT) language, perform this function. For example, we may have one XSLT that reformats an XML message as a Web page, another that outputs a plain-text document for printing, and a third that outputs data formatted as expected by a legacy application. To summarize, we use XML to "tag" content in a message, XSD to define the structure of the tags, and XSLT to reorganize the data based on the needs of a specific consumer.

C. SERVICES

"A service is an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes." (O'Brien, Bass, and Merson 1) Web services differ from generic services because they use SOAP-formatted XML envelopes and have their interfaces described by a Web Service Description Language (WSDL) document. Section D defines both SOAP and WSDL. We use the terms service and Web service interchangeably throughout this thesis. The decision to use one or the other will be made by the architecture team for each service, when it designs the SOA.

1. Common Principles of Service Orientation

The authors of a recent Software Engineering Institute report on SOA provide the following service oriented design principles. They establish a unique design approach for building Web services for SOA. "When applied, these principles succeed in standardizing Web services while preserving their loosely coupled relationships." (Erl 53)

> Services are reusable. Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.
>
> Services share a formal contract. In order for them to interact, they need not share anything but a formal contract that defines the terms of information exchange and any supplemental service description information.
>
> Services are loosely coupled. They must be designed to interact on a loosely coupled basis, and they must maintain this state of loose coupling. This is closely related to service abstraction and service autonomy. [*Loosely coupled frameworks allow individual nodes in a distributed system to change without affecting or requiring change in any other part of the system.*]
>
> Services abstract underlying logic. The only part of a service that is visible to the outside world is what is exposed via the service's description and formal contract. The underlying logic (beyond what is expressed in the description and formal contract) is invisible and irrelevant to service requestors.
>
> Services are composable. They may compose other services. This possibility allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.
>
> Services are autonomous. The logic governed by a service resides within an explicit boundary. The service has complete autonomy within this boundary and is not dependent on other services for the execution of this governance.
>
> Services are stateless. They should not be required to manage state information, since that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
>
> Services are discoverable. They should allow their descriptions to be discovered and understood by humans and service users who may be able to make use of the services' logic. Service discovery can be facilitated by the use of a directory provider, or, if the

> address of the service is known during implementation, the address can be hard-coded into the user's software during implementation.
>
> Services have a network-addressable interface. Service requestors must be able to invoke a service across the network. When a service user and service provider are on the same machine, it may be possible to access the service through a local interface and not through the network. However, the service must also support remote requests.
>
> Services are location transparent. Service requestors do not have to access a service using its absolute network address. Requestors dynamically discover the location of a service looking up a registry. This feature allows services to move from one location to another without affecting the requestors. (O'Brien, Bass, and Merson 3-4)

2. Wrappers

Architects often want to reuse existing applications and databases in their SOAs. Unfortunately almost all legacy systems cannot operate in the service environment in their current configuration. Developers solve this problem by creating a wrapper, special software that resides between the legacy application and the SOA. The wrapper exposes the legacy application's functionality or data to the SOA as a service. The wrapper provides all the security, quality of service, and service orientation principles that any other service in the SOA has. The following quote illustrates the benefits wrappers can provide:

> For example, at telecom company Verizon, the service called "get CSR" (get customer service record) is a complex jumble of software actions and data extractions that uses Verizon's integration infrastructure to access more than 25 systems in as many as four data centers across the country. Before building the "get CSR" service, Verizon developers who needed that critical lump of data would have to build links to all 25 systems—adding their own links on top of the complex web of links already hanging off the popular systems. But with the "get CSR" service sitting in a central repository on Verizon's intranet, those developers can now use the simple object access protocol (SOAP) to build a single link to the carefully crafted interface that wraps around the service. Those 25 systems immediately line up and march, sending customer information to the new application and saving developers months,

even years, of development time each time they use the service. ("ABCs of SOA")

Wrappers provide an excellent way to reuse applications already delivering business value. However, proper IT business alignment is necessary to ensure proper enforcement of control and management policies. Randomly wrapping services can lead to security and performance problems inside and outside the organization. The Web service wrappers provide a "great tactical approach" for SOA development, but they are not a panacea. ("Web Services Wrapper") We can not simply wrap all our legacy systems and declare SOA victory. Ultimately, SOA aims to unlock the application logic and data from the legacy systems, so they exist as native services within the SOA. This process frees them to operate at their logical place in the business processes and workflows, without the artificial constraints of the legacy systems.

D. WEB SERVICE STACK

The Web services stack shows the collection of computer networking protocols that define, locate, implement, and make Web services interact with each other. The World Wide Web Consortium's Web Services Architecture Working Group defined technical standards to ensure interoperability for SOAs. The Working Group divided these standards into the following six areas: processes, descriptions, messages, communications, security and management: Figure 3 shows a modified version of their Web Services Architecture Stack diagram.

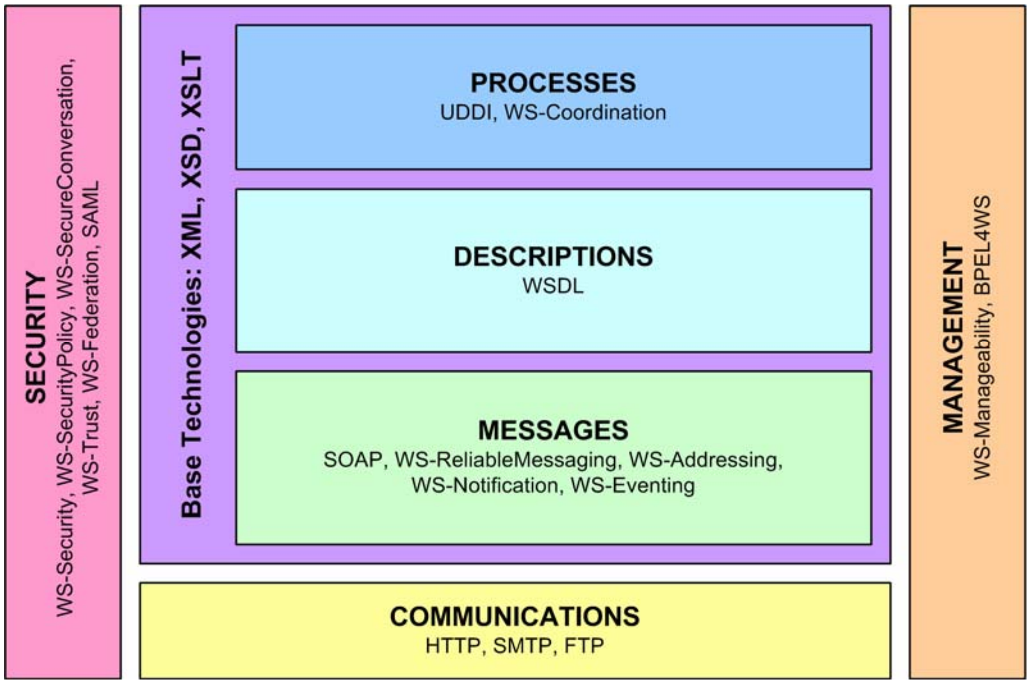

SECURITY
WS-Security, WS-SecurityPolicy, WS-SecureConversation, WS-Trust, WS-Federation, SAML
Base Technologies: XML, XSD, XSLT
PROCESSES
UDDI, WS-Coordination
DESCRIPTIONS
WSDL
MESSAGES
SOAP, WS-ReliableMessaging, WS-Addressing, WS-Notification, WS-Eventing
COMMUNICATIONS
HTTP, SMTP, FTP
MANAGEMENT
WS-Manageability, BPEL4WS

Figure 3. Web Services Architecture Stack (After "Web Services Architecture" Figure 3-1)

1. Process Layer

The Process layer describes how providers publish services and requestors/consumers discover them. The Process layer utilizes the following standards:

- Universal Description Discovery and Integration (UDDI): UDDI is a directory that allows businesses to register their services so that the consumers can find them.
- WS-Coordination: This specification "describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed activities." ("WS-Coordination")

2. Description Layer

The Description layer describes how the service provider communicates the specifications for invoking the Web service to the service requestor. The Description layer utilizes the following standards:

- Web Service Description Language (WSDL): An XML document that describes the interfaces and methods that a service provides.

3. Messages Layer

The Messages layer describes how the services pass information in the form of a message. The Messages layer utilizes the following standards:

- Simple Object Access Protocol (SOAP): SOAP is a protocol used to exchange messages between systems in XML format. SOAP has become the de-facto standard protocol for Web services.
- WS-ReliableMessaging: This specification describes a protocol that allows messages to be transferred reliably between nodes in the presence of software component, system, or network failures. (“WS-ReliableMessaging”)
- WS-Addressing: This specification “provides transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.” (“WS-Addressing”)
- WS-Notification: “The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example in publish/subscribe systems provided by Message Oriented Middleware vendors, or in system and device management domains.” (“WS-Notification”)
- WS-Eventing: “This specification describes a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages.” (“WS-Eventing”)

4. Communications Layer

The Communications layer describes how messages are physically transported across the network. The Communications layer utilizes the following Internet protocols:

- Hypertext Transfer Protocol (HTTP): HTTP is the standard mechanism for retrieving Web pages and associated content. It can also be used for transmitting data from the client to the server.
- Simple Mail Transfer Protocol (SMTP): SMTP is the standard mechanism for sending email from the client to the server.

- File Transfer Protocol (FTP): FTP is primarily used for transferring files from one computer to another over a TCP/IP network.

5. Security

Security occurs at all layers in the stack and it provides authenticity, integrity, confidentiality, and non-repudiation. Security utilizes the following standards:

- WS-Security: "This specification describes enhancements to SOAP messaging to provide message integrity and confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies." ("WS-Security")
- WS-SecurityPolicy: WS-SecurityPolicy is designed to work with the general Web Services framework including WSDL service descriptions, UDDI businessServices and bindingTemplates and SOAP message structure and message processing model, and WS-SecurityPolicy should be applicable to any version of SOAP. ("WS-SecurityPolicy")
- WS-SecureConversation: "This specification defines extensions that build on WS-Security to provide a framework for requesting and issuing security tokens, and to broker trust relationships." ("WS-SecureConversation")
- WS-Trust: The goal of WS-Trust is to enable applications to construct trusted SOAP message exchanges. This trust is represented through the exchange and brokering of security tokens. This specification provides a protocol agnostic way to issue, renew, and validate these security tokens. ("WS-Trust")
- WS-Federation: A specification, by IBM and Microsoft, for standardizing the way companies share user and machine identities among disparate authentication and authorization systems spread across corporate boundaries. ("WS-Federation")
- SAML: "An XML-based framework for communicating user authentication, entitlement, and attribute information. As its name suggests, SAML allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject (an entity that is often a human user) to other entities, such as a partner company or another enterprise application." ("SAML")

6. Management

Management, like Security, occurs across all layers in the stack. Management provides methods for monitoring and managing services and business processes. Management utilizes the following standards:

- WS-Manageability: "specification introduces the general concepts of a manageability model in terms of manageability topics and the aspects used to define them." ("WS-Manageability")
- Business Process Execution Language for Web Services (BPEL4WS): "The Business Process Execution Language for Web Services provides a comprehensive syntax for describing business workflow logic. It allows for the creation of abstract processes that can describe business protocols, as well as executable processes that can be compiled into runtime scripts" (Erl 100) The Business Process Modeling Notation (BPMN) provides a standardized graphical notation for drawing business processes in a workflow. Software tools easily translate BMPN models into BPEL4WS files.

E. DATA MODELS AND INFORMATION EXCHANGE

Semantics define a language's structure and meaning. For the services in an SOA to be interoperable, the services exchanging messages must understand the semantics of the data. Therefore, we need an efficient way to establish an agreed upon structure and meaning for the data elements in our XML formatted messages. Conceptual data models and XML schemas accomplish this.

Conceptual data models (CDM) show the overall organizational data structure without considering the ability to implement the structure. SOAs employ CDMs to avoid the point-to-point mapping problem encountered when sharing data between systems. For example, if we have N systems in our SOA and we want them all to share data with each other, point-to-point mapping requires N(N-1) translations between them which is approximately N^2. Utilizing a CDM requires 2N translations; one for each system to the CDM and one for the CDM back to each system, and this number is usually much smaller than N^2. Interestingly, Appendix D to the Coast Guard's Common Operational Picture Operational Requirements Document (COPORD) shows a matrix proposing point-to-point mapping between nine existing stovepipe systems. The 72 transformations required in the diagram could be reduced to 18, a reduction of 75%, through appropriate use of a CDM. The Coast Guard has not yet implemented a command and control CDM.

Creating a CDM has one negative aspect. The data modeling and related XML schema generation efforts increase the start-up cost. XML schemas describe an XML document's structure and validate messages in the SOA. "This industry best practice requires work up front, but results in a scalable and flexible solution. The instantiation of a canonical XML Schema based on that model provides a consistent target … to which each endpoint system maps." (Hutchins) Conceptual data modeling benefits outweigh their costs, in a way similar to the payback-to-cost provided by the Incremental Evolutionary approach in Chapter 4's Figure 22.

F. EXAMPLE WEB SERVICE

1. Search Pattern Service (SPS) Description

We created the SPS to provide an example. It accepts search pattern parameters and returns the latitude and longitude points for the waypoints along the search. Coast Guard readers may initially dismiss the need for a service to generate search patterns. We already have many different systems capable of doing this, and several provide much more robust functionality. However, in addition to illustrating what a service can do, this particular example highlights a more important point. The Coast Guard has recreated the same functionality in a dozen different systems, but we can't take a search pattern generated by the Sector Command Duty Officer (CDO) and automatically import it into the navigation system on a patrol boat.

Implementing functionality as a service means that you only need to build it once. Thereafter, anyone can use it anywhere in the SOA. Service modifications only happen in one location, not in each separate system. The SPS can also have multiple interfaces so that many different applications and devices can use it. This service calculates the same search pattern coordinates for every user. The Sector CDO sees the exact same pattern at his computer workstation that the coxswain on the small boat sees on his SINS equipment. The C-130 sees the same pattern on his Cockpit Display Navigational Unit (CDNU) as the District Commander using a smart phone. However, perhaps we don't want it to calculate the same pattern for every user. The interface can also make the service context-sensitive. For example, the small boat interface would generate search patterns that avoided rocky shoals but the helicopter interface would not.

2. Current Features

The SPS accepts the parameters shown in Table 1 then calculates and returns a series of latitude/longitude pairs. Appendix C contains the SPS source code. It currently performs three search patterns:

- Parallel Search
- Sector Search
- Expanding Square Search

Search Pattern	Parallel	Sector	Expanding Square
Latitude	x	x	x
Longitude	x	x	x
Length	x		
Width	x		
Track Spacing	x		x
Major Axis	x		
Radius		x	
Theta		x	
Initial Track		x	x
Cycles			x

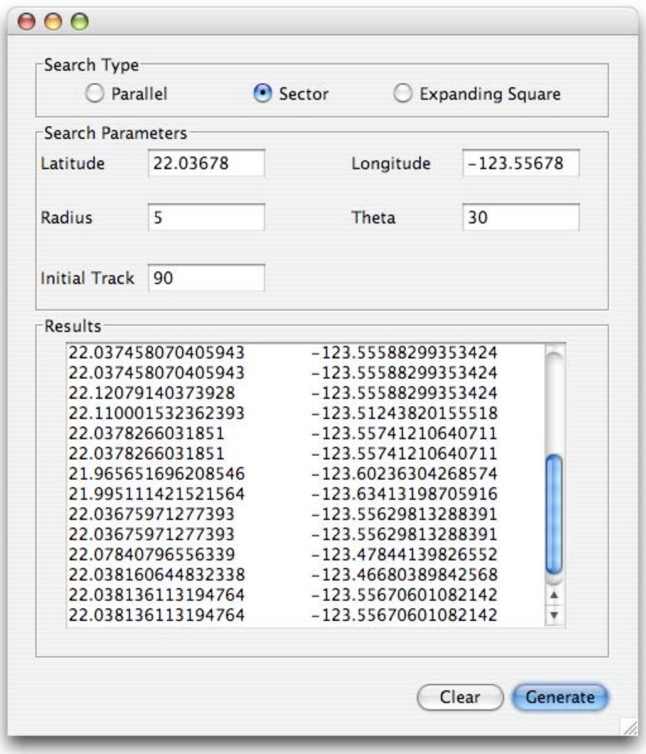
Table 1. Search Pattern Service Parameters

3. Potential Future Features

The initial description mentioned one potential feature, adjusting the pattern based on the asset type. It could also access current weather and sea conditions, or receive updated information about the target, and then dynamically adjust the search pattern parameters accordingly. Couple this capability with the integrated navigation systems in some of the Coast Guard's assets and it becomes possible to improve mission effectiveness. Currently, changes to search patterns require the coxswain or pilot to manually stop the current search and enter a new one with the updated information. Dynamic updates allow the people performing the mission to keep their eyes and attention focused on finding the survivor rather than buried in a navigation computer entering new coordinates.

4. Java-based Client Application

We created a multi-platform Java-based client to demonstrate this service's functionality and output. Appendix D contains the Java client source code. The user enters parameters into the form boxes and initiates the service by pressing the "Generate" button. This client initiates a request to the service and displays the result in the "Results" text box. Figure 4 shows the Java client in the sector search mode. Note that the latitude and longitude coordinates are in degrees with decimals. If we needed positions in degrees, minutes, seconds that conversion could be built into either the client application or the service itself. This client would probably not exist within the Coast Guard SOA. Most services do not need a dedicated user client. Integrated user interfaces, called composite applications, fuse the functionality and output of many services.



Search Type

Parallel [✓ Sector] Expanding Square

Search Parameters

Latitude	22.03678	Longitude	-123.55678
Radius	5	Theta	30
Initial Track	90		

Results

```
22.037458070405943        -123.55588299353424
22.037458070405943        -123.55588299353424
22.12079140373928         -123.55588299353424
22.110001532362393        -123.51243820155518
22.0378266031851          -123.55741210640711
22.0378266031851          -123.55741210640711
21.965651696208546        -123.60236304268574
21.995111421521564        -123.63413198705916
22.03675971277393         -123.55629813288391
22.03675971277393         -123.55629813288391
22.07840796556339         -123.47844139826552
22.038160644832338        -123.46680389842568
22.038136113194764        -123.55670601082142
22.038136113194764        -123.55670601082142
```

Figure 4. Java Client – Sector Search

G. CONCLUSION

This chapter defined important terms, introduced relevant concepts, identified applicable industry standards, and summarized SOA principles. Designing and building an SOA requires appropriate data standardization and modeling into XML schemas, adhering to the important industry standards to appropriately implement the technology needed to support the layers of the Web services stack. The next chapter defines an SOA for Coast Guard command and control to answer our first thesis question.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DRAFT USCG COMMAND AND CONTROL SERVICE ORIENTED ARCHITECTURE (CGC2 SOA)

A. INTRODUCTION

This chapter provides the answer to our first thesis question, "How can the Coast Guard implement a Service-Oriented Architecture for Command and Control?" We begin with three diagrams in Section B; one to introduce the functional areas, one to describe the network nodes, and one to illustrate the systems interfaces. We then continue with a scenario in Section C that demonstrates the CGC2 SOA in action. Section D further describes the functional areas and identifies services to perform them. Section E discusses the quality attributes that significantly influence the architecture. Section F outlines the conceptual data model required to exchange data within the SOA. Finally, Section G proposes a product line architecture for producing composite applications. The answer to our first thesis question does not create a complete architecture. However it does establish an effective starting point for the Coast Guard.

B. ARCHITECTURAL VIEWS

"Software architecture represents a common abstraction of a system that stakeholders can use as a basis for creating mutual understanding, forming consensus, and communicating with each other." (Clements, Kazman, Klien, 2) Diagrams can elegantly summarize complex material, clearly showing details that otherwise get lost in lengthy text descriptions. Architectural views are diagrams that provide a mechanism for separating issues and concerns when analyzing or building an architecture. "They let us consider an architecture from different perspectives." (Clements, Kazman, Klien, 8) The Department of Defense (DoD) uses a framework that "defines a common approach for DoD architecture description, development, presentation, and integration" called the DoD Architecture Framework. (*DoDAF Deskbook* 1-1) We created our diagrams as DoDAF "views" in order to facilitate comparison between our CGC2 SOA and existing DoD and Coast Guard command and control systems. The following diagrams describe our proposed CGC2 SOA.

1. High Level Operational Concept (OV-1)

The OV-1 displays the primary CGC2 SOA actors. Participants include Coast Guard units, as well as federal, state and local governments and non-governmental agencies (e.g., harbor pilot associations and shipping companies). The five small ovals in Figure 5 represent the CGC2 SOA functional areas. The large blue oval represents the combined command and control effect those functions produce, making the whole greater than the sum of the parts.

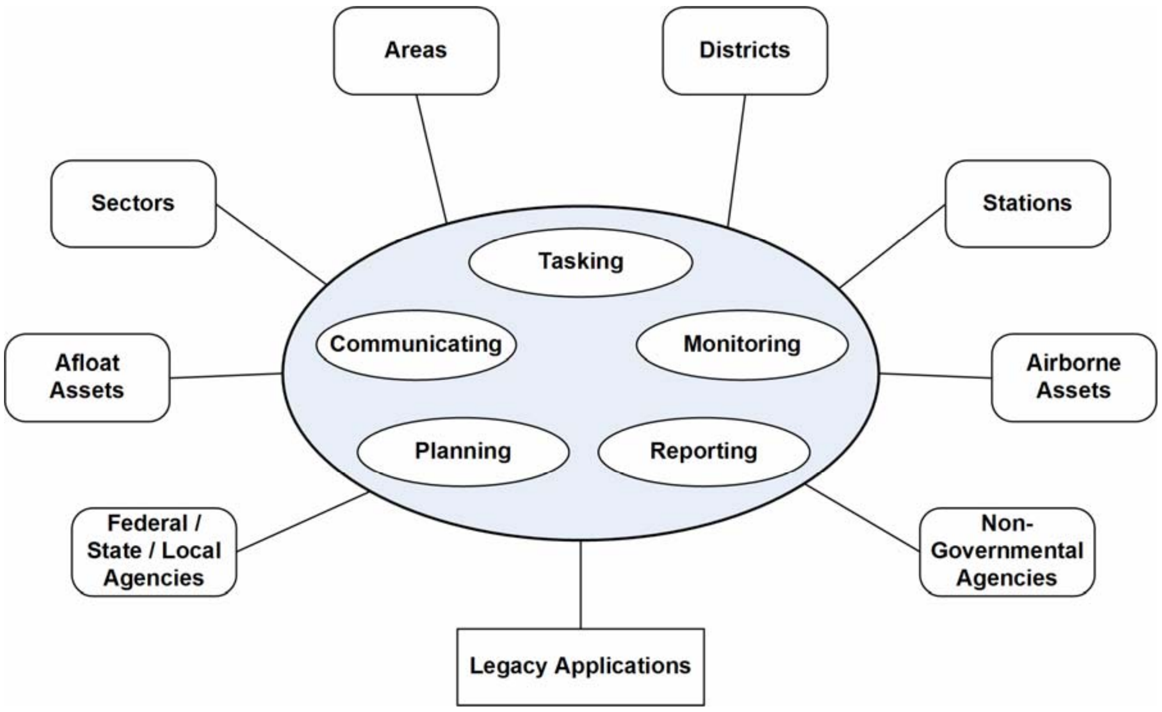


Figure 5. CGC2 SOA Functional Areas and Actors (OV-1)

2. Operational Node Connectivity Description (OV-2)

The OV-2 shows all nodes that use, produce and consume information from services throughout the organization. These services send and receive messages formatted in the Extensible Markup Language (XML). The existing Coast Guard Data Network (CGDN) provides connectivity between network nodes, but does not reach mobile assets (e.g., aircraft and small boats).

Appendix B lists our proposed CGDN and communications systems requirements.

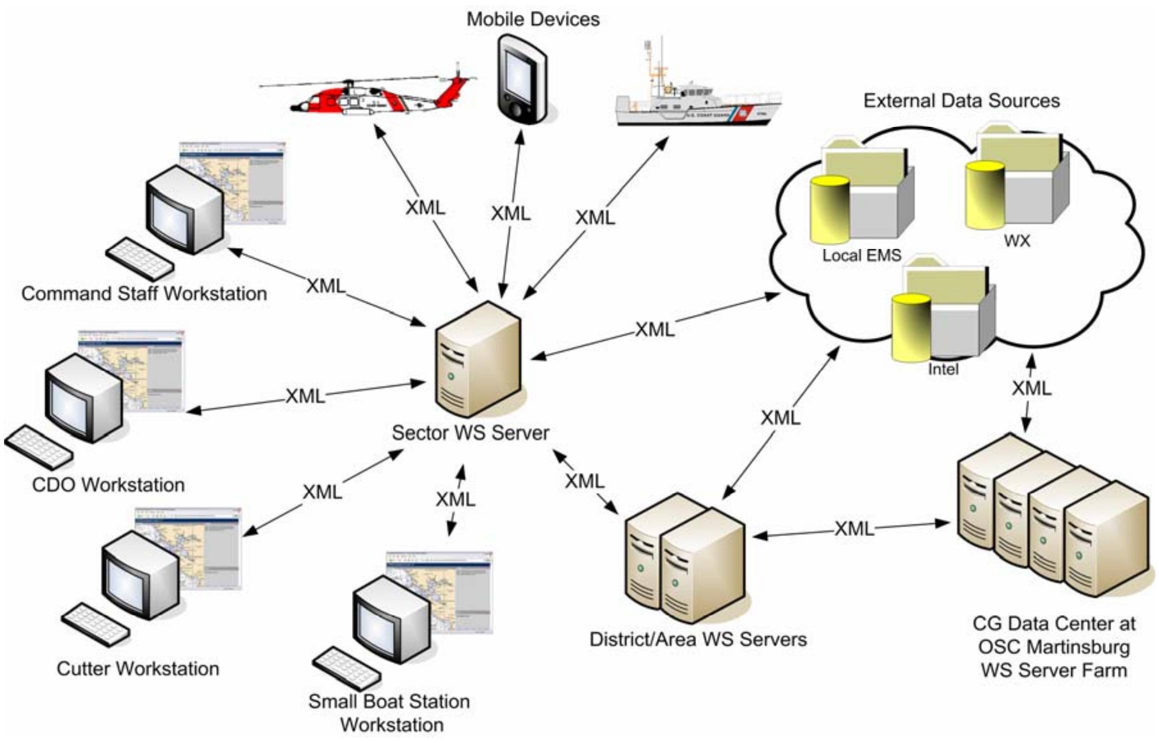


Figure 6. CGC2 SOA Operational Node Connectivity (OV-2)

3. Systems Interface Description (SV-1)

The SV-1 identifies the interfaces between systems and system nodes. The diagram in Figure 7 shows the relationship between legacy systems, elemental and composed services, and the composite applications that utilize them. The gray horizontal boxes abstract many complex implementation details. Although each legacy system has unique requirements, SOA allows service providers and consumers to utilize any technology that supports the appropriate standards. While extremely important, these non-architectural issues will not be addressed in this thesis.

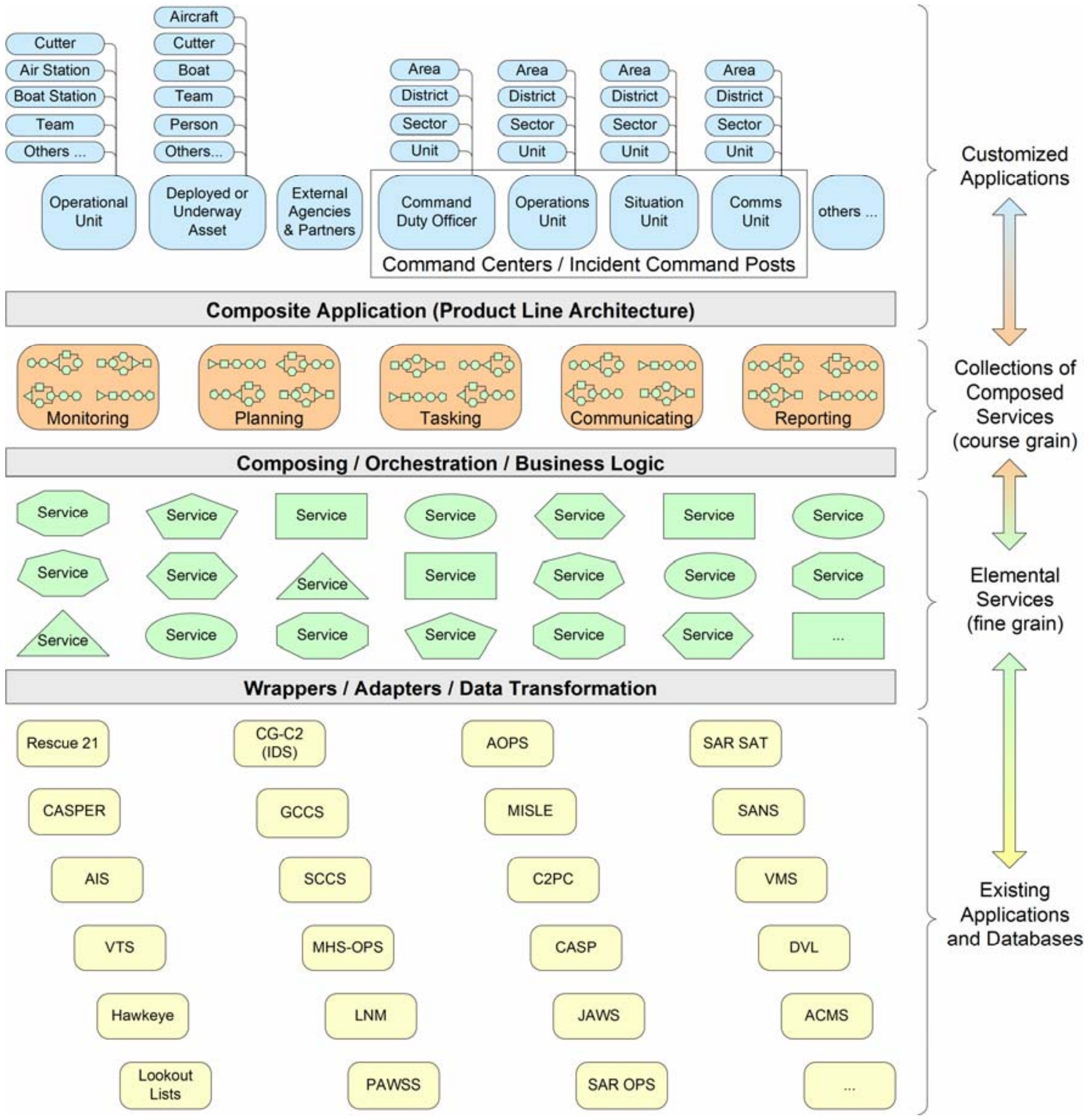


Figure 7. CGC2 SOA Systems Interface Description (SV-1)

The yellow boxes at the bottom of Figure 7 represent legacy applications that must be "service-enabled." To do this, software called wrappers change the existing applications' interfaces without affecting current functionality. Wrappers expose the business logic and data from legacy applications as services, which can be invoked (used) within the SOA. The wrappers also perform data transformation between the legacy application and the SOA's context data model, which will be explained in Section G. Wrapping multiple legacy applications creates a pool of fine-grained services. The green shapes in Figure 7 represent all the fine-grained services. Each one typically performs a single business logic or data access function. Five different fine-grained service examples are: get local assets (e.g., cutters, boats, aircraft) currently in Alpha or Bravo status, get asset positions, get OPAREA weather, filter asset list based on weather limits, and sort asset list based on distance from present location to target.

As they pass through the middle gray box, the fine-grained services are assembled into processes, or workflows, that perform complex business functions. These processes or assemblies are known as coarse-grained services. The orange boxes in Figure 7 represent the five functional areas that logically group the course-grained services. The fine-grained services described in the paragraph above could be linked together to form a “nominate asset” service under Planning. This service would take a geographic position, perform those fine-grained services, and return a list containing available assets. Changing the fine-grained services’ input parameters can customize this coarse-grained service. For example, the “nominate surface” service would include cutters, boats, and Automated Merchant Vessel Reporting (AMVER) vessels, but the “nominate air” service would only return aircraft. Items from the coarse-grained services inventory can be reused as needed anywhere in the CGC2 SOA. In this way, we build functionality once, and then quickly deploy it across all units and mission areas.

The typical user interacts with the services through a composite application, shown as blue boxes at the top of Figure 7. A Command Duty Officer (CDO) needs a different composite application than the Sector Commander, which will differ from that needed by the district staff officer. Therefore we want an adaptive and inexpensive way to create a composite application tailored for each user type. We propose a product line architecture to create these composite applications in Section G. This will allow the Coast

Guard to quickly create multiple composite application variations needed by different command and control user groups.

Now that we've established how the logic and data from existing legacy applications is organized (OV-1), distributed (OV-2), and consumed (SV-1), we will bring it all together in Section C with a scenario showing the CGC2 SOA in action.

C. SCENARIO

The following scenario illustrates several CGC2 SOA services, marked with [*service name*]. Throughout these events, the Sector Command Center watchstanders use electronic checklists, linked to tasking and communicating services that prompt users to perform required actions and automate information dissemination. The system records each service action in it's log files. In addition, watchstanders select specific actions to insert into their standard Coast Guard logs, now kept in electronic form.

10 January 2008 – [*View Plan*]: Sector San Francisco's Response Department staff reviews the Quarterly Operations Schedule for events during the upcoming week. The entire weekly schedule must be reviewed because several maintenance and training plans have changed from the time when the quarter schedule was created. [*Monitor Request-Pull: SANS*]: The staff also accesses the local vessel arrival notices to determine Homeland Security boarding and escort activities. [*Create Plan*]: The staff creates a weekly schedule that balances competing demands for operational assets. [*Create Task*]: They create detailed tasks that include patrol areas and relevant available information about the target vessels, cargo and crew members. [*Approve Plan*]: The Sector's command staff electronically reviews the Weekly Schedule and approves it. [*Assign Plan, Assign Tasks, Send Message*]: The approval triggers the system to assign the plan and associated tasks to all units. [*View Plan, View Task*]: Any authorized user can access the approved schedule and associated tasks.

16 January 2008 –The Rescue 21 system detects a Mayday call. [*Monitor Receive-Push: Rescue21DF*]: It generates an alert that includes an estimated position (triangulated from direction finding antennas) and the Mayday call's digital recording. The communications watchstander unsuccessfully attempts to hail the vessel on the radio. [*Create Case, Modify Case*]: The CDO creates a SAR case and appends the alert to the case file. The Sector Command Center watchstanders listen to the digital recording. The position from the Rescue 21 alert does not correspond to what the person says during the Mayday call. Another fishing vessel radios to report they overhead the distress call. This vessel reports that the Mayday came from the "LUCKY LADY", and that they heard the victim say the vessel had two people on board. [*Modify Case*]: The CDO adds this information to the case file. [*Report Request*]: The CDO queries available databases for information about vessels with the name "LUCKY LADY." [*Monitor Receive: SARSAT*]: Shortly thereafter the Sector receives a SAR Satellite (SARSAT) alert from an unregistered Emergency Position Indicating Radiobeacon (EPIRB) reporting a position 0.5 nautical miles from the Rescue 21 alert position. [*Report Generator*]: The CDO receives a report back from the database query listing 3 vessels within the Sector San Francisco area of responsibility. The system matches the registration number from the EPIRB alert to a vessel in the database report. [*Modify Case*]: The CDO inserts the SARSAT alert and matching vessel record from the database query into the case file. The CDO does the same for all subsequent information related to the case.

The SAR case creation triggers the nomination of available assets. [*Available Assets*]: The Sector CDO receives a list with aircraft, boats, and cutters. Assets marked green have the appropriate readiness level and ability to operate in the forecasted environmental conditions. The remaining assets are marked in yellow or red. [*Assign Case*]: The CDO selects an HH-65 helicopter (6501) from AIRSTA San Francisco and a 41' boat (41001) from Station Golden Gate. [*Create Message, Send Message*]: The SAR case and associated information is sent to both responding units. [*Create Report, Create Message,* *Send Message*]: The CDO sends the Sector commander and staff a summary report with a hyper-link allowing recipients to view the SAR case file.

[*Search Pattern*]: The system generates search patterns based on the reported positions, selected assets, vessel-in-distress size and type, number of suspected people in the water, and the forecasted weather. [*CASP*]: These proposed search patterns include probability of detection information that allows the CDO to verify their appropriateness. [*Create Task*]: The CDO creates specific tasks for the 6501 and 41001 to execute these search patterns. [*Assign Task, Create Message, Send Message*]: The CDO sends search pattern tasks to the responding units in a format that automatically loads into their navigation display systems. The CDO sends a summary report to the Sector commander and staff.

[*Database Query: AOPS*]: 6501 and 41001 dispatch to perform their assigned tasks and their change in status is automatically recorded. [*Monitor Receive-Push: Blue Force Tracker*]: Throughout the following events, the CDO receives helicopter and boat position and status information. The helicopter arrives on scene with the vessel, lowers a pump to the vessel, and recovers one person from the water. [*Create Message, Send Message*]: 6501 sends patient data to Sector San Francisco. [*Create Message, Send Message*]: 6501 departs to take the victim to a nearby hospital and the CDO forwards the available patient data to the local emergency medical services. [*Create Message, Send Message*]: The CDO sends updated target vessel position information to 41001 during its transit from the station to the scene. [*Modify Case*]: 41001 locates the vessel with the remaining person onboard and tows it back to port. The CDO marks the SAR case complete. [*Report Request, Report Aggregator, Report Generator*]: This action triggers several reports, including one that automatically initiates and populates the reports required from the small boat and aircraft with the case file information.

<u>17 Jan 2008</u> – [*eNOAD*]: a container ship, M/V OCEAN TRADER, scheduled to arrive at 2300 submits an updated Advanced Notice of

Arrival/Departure reporting they have been delayed 18 hours. [*Monitor Receive-Push: eNOAD, Task Schedule Check*]: Sector San Francisco receives the report from the National Vessel Movement Center. [*Create Message, Send Message*]: This ship was already scheduled for boarding and escort into port based on irregularities in the cargo manifest. Sector San Francisco originally assigned this task to USCGC TERN and a boarding team from the Vessel Boarding and Search Team (VBST). [*Receive Message*]: The CDO receives the alert message indicating that OCEAN TRADER's delay impacts an assigned task. The alert shows a schedule conflict between the new boarding time and TERN's dockside maintenance period. It lists two alternatives for the escort duty, USCGC PIKE and 41010 from Station San Francisco. [*Assign Task, Create Message, Send Message*]: The CDO selects PIKE and the service automatically reassigns the tasks. The service updates the VBST's task to reflect the new cutter assignment. [*Monitor Receive-Push: AIS*]: Based on the task assignment, PIKE receives OCEAN TRADER's position information from the Automated Identification System.

D. FUNCTIONAL AREAS

The actions (operations) and capabilities performed by the system for the user defines the systems' functionality. We divide our SOA's functionality into five different areas as indicated in Figure 5 above. This section describes these areas in detail, identifies the legacy stovepipe systems, and defines a small portion of the services required to implement the architecture.

1. Planning

The planning area encompasses all mission planning, event scheduling, and resource allocation functions. It corresponds to the "planning" capability in the Command Center Program Manual (CCPM). It includes deliberate planning operations, and crisis action planning for emergent events such as search and rescue, marine environmental protection response and disaster response.

a. Discussion

The planning area's base services and data model must be carefully constructed. They must contain enough details to meet individual planner's needs at all levels, without including so many details as to become cumbersome and unmanageable. Given the wide range of missions and scope of operations, all planners do not have the same needs. Users at each level need the functionality and appropriate user interface for their mission area. Some information applies to all missions, but individual communities will extend the basic structure with details specific to their requirements. The challenge lies in ensuring each planner has the details they need to effectively plan their mission, without overwhelming the system.

The following deliberate planning cycle example illustrates the needs of Coast Guard planners. At the strategic level, the area command staff promulgates annual goals and targets. Each district takes those goals and creates the operational plan for their subordinate sectors. The sector staff turns those planning goals into tactical missions assigned to individual response units. Each unit creates a unit level plan to assign resources and personnel for each mission based on their personnel and equipment readiness. This multi-level process happens in each mission area, for each iteration of the deliberate planning cycle.

This process would be relatively straightforward if we only considered law enforcement, vessel safety inspection, or any one individual mission. It becomes much more complex when we expand the planning needs at each step in the process to 10 or 20 different missions. The staff at a sector continuously plans for law enforcement, homeland security, vessel safety, and port state compliance operations, just to name a few. While these operations share some common planning details, each mission area does have unique content. Sector command centers additionally need to perform crisis action planning to respond to SAR, marine accidents, and pollution incidents. One rigid system will not meet the planning needs for all missions at all organizational levels.

The Coast Guard needs a well balanced, component-based planning system that can provide and integrate tailored solutions specialized for different users and missions. Consider the scenario in Section C above. In the CGC2 SOA planning services, efficiency and accuracy prevail up and down the chain-of-command, in stark contrast to current stovepipe planning procedures. As the typical deliberate planning cycle currently happens, planners have their own system to document and pass information at each level. Strategic guidance lives in Microsoft Word documents or message traffic. Operational planners create a local document, spreadsheet or small database to manage their data. Often, they email these files to subordinate units or place them in a public folder on a shared server. Each unit then creates a local system for managing their tactical scheduling, as well as other local functions. This inherently inefficient procedure limits our ability to respond in the dynamic environment. Status changes made at the unit level (as when the USCGC TERN was no longer available in the scenario) do not necessarily get reported back up the chain. A well constructed planning system will give decision makers at all levels the most accurate and timely information to make intelligent decisions. Improved decision making enables us to better serve our customers and more effectively use our scarce resources.

b. Legacy Planning Systems

The following legacy planning systems can provide functionality in the CGC2 SOA:

- Maritime Homeland Security Operational Planning System (MHS-OPS): a homeland security operational and tactical mission planning and scheduling application.
- Computer Aided Search Planning (CASP): a SAR planning tool used to determine search object drift, over a defined time, in an off shore oceanic environment.
- Joint Automated Worksheet (JAWS): a SAR planning tool used to determine search object drift over time and calculate optimal search areas utilizing available assets.
- Search and Rescue Optimal Planning System (SAR OPS): a SAR planning tool that uses environmental data to develop optimal search plans based on a defined “effort”, or resource hours available.

c. Planning Services

The following descriptions provide a service framework to meet the planning needs described above. The first six define general tasking services we will reuse in other Coast Guard SOAs:

- Create Plan: Create and populate a new plan. This includes options for several different plan types including JOPES, ICS, Annual Schedule, Quarterly Schedule, and Weekly Schedule.
- Modify Plan: Changes data elements in existing plans. This service also appends any supporting tasks to the plan.
- View Plan: Displays existing plans in various formats. This service may be called by the reporting services.
- Validate Plan: Error and omission check.
- Approve Plan: Tracks a plan's review and approval by the chain of command.
- Assign Plan: Assign a plan to a subordinate unit.
- Available Assets: Generates a list of assets in Bravo or Alpha status, located within the response range of a given geographic position at a given time.
- CASP Service: Generates search pattern "probability of detection" graphic.

A case contains information about a specific Coast Guard mission event. Any mission area can create a case, however law enforcement, SAR, and Marine Safety events typically initiate them. Cases also store intelligence information related to a specific event or entity (vessel, person, cargo, facility, company). We discuss cases in the planning section because planning typically happens in conjunction with a case being created. Cases also fit in the reporting section although we do not discuss them there.

- Create Case: Create and populate a new case. This includes options for several different case types, like Search and Rescue, Law Enforcement, and Marine Investigation.
- Modify Case: Makes changes to data elements in existing cases. This service also appends supporting information and electronic documentation (evidence) to the case.

- View Case: Displays existing cases in various formats. Reporting services may call this service.
- Assign Case: Assign a case to a subordinate unit.

d. Planning Conclusion

The Coast Guard has taken a step in the right direction with the prototype MHS-OPS system. It implements some of the planning functionality described above, and this shows promise. However, in typical Coast Guard fashion, we limited it to only meet the needs of one mission. This system does offer an excellent entry point for building our CGC2 SOA's planning functions. To begin with, MHS-OPS must be service-enabled and integrated into the SOA.

2. Tasking

Once we have planned our operations, the services in the tasking functional area must enable effective resource assignment. For this discussion, we define tasking as the point in the process where a decision-maker directs a specific asset to go to an assigned point to perform a particular objective. Each plan usually includes multiple tasks.

a. Discussion

Within the Coast Guard "we push both authority and responsibility to the lowest possible level. Our ethos is that the person on scene can be depended upon to assess the situation, seize the initiative, and take the action necessary for success." (*America's Maritime Guardian* 52) This organizational culture stems from "Coasties" ,operating without constant communication with their superiors over the last two-hundred seventeen years. We still embrace this autonomous operational environment today. Operators use their commander's stated goals and applicable Coast Guard policies as the basis for their on-scene decisions. Commanders expect situations to change as assets operate in a dynamic environment. Therefore, our tasking services will focus on telling an asset to "go and do" without encumbering them with overly complex task descriptions.

The services within this functional area provide the following information: asset being tasked, user assigning task, action to perform, place, time, and target description. Section F describes this data structure. The place and target aspects provide a dramatic improvement over existing capabilities. The place information includes track-lines, search patterns, and geographic points. The target information contains sufficient detail for the asset to locate and identify the object, including description and tracking information from all connected systems. The ability to electronically transmit and then automatically display and utilize information from other sensors and systems will significantly improve Coast Guard command and control.

b. Legacy Tasking Systems

The following legacy tasking system can provide functionality in the CGC2 SOA:

- Incident Command System (ICS): a standardized national response management system used during crisis and non-crisis events.

c. Tasking Services

The following descriptions provide the framework for creating services to meet the tasking needs described above. The first four were written as general tasking services that we will reuse in other Coast Guard SOAs.

- Create Task: Create a new task.
- Modify Task: Modify existing tasks.
- View Task: View existing tasks. The reporting module services can also call these services..
- Assign Task: Assign a task to a subordinate unit.
- Scheduled Task Check: Compares existing task requirements with the associated asset's current or scheduled readiness condition. A conflict triggers the Available Asset service to prompt the CDO with a replacement candidate list.
- Search Pattern: Generates positions for a search pattern based on standard inputs, see Chapter 2 for more details.
- Environmental Limits Check: Accepts weather data and asset type and compares the asset's operational limits to the forecasted weather. It returns a "go/no go" recommendation for each asset, including the exceeded limits that cause a "no go"

- Create Target: Compiles position and other related track information for transmission to an asset. Provides several output formats to meet different asset navigation and display systems' needs.
- Intercept Target: Receives information from "Create Target" service and generates local intercept solution displayed on the assets navigation or display system. An enhanced version fuses the external target data with the asset's organic sensors to produce a more accurate intercept solution.

d. Tasking Conclusion

As with the Planning services, the base Tasking services need to address the principal mission area details. We will develop additional services to provide the unique functionality required in each mission area.

3. Communicating

The communication functional area provides the SOA's backbone essential to the system's success. The best planning and tasking in the world accomplishes nothing if no one knows about it. The communications services will generate and disseminate many routine information alerts as well as enable real-time and asynchronous communication between personnel and systems. At the system level, it will conduct messaging between services. Services generate messages automatically and invisibly to most users, but these important messages implement the planning and tasking functions already discussed. The following paragraphs highlight some differences between personal messaging and service messaging.

a. Discussion

Service messaging in the SOA requires no user action, which enables great efficiency and performance. Removing the human element from routine monitoring, data fusion and transmission increases communications quality and timeliness. Consider the HLS escort and boarding scenario without SOA. The M/V Ocean Trader updates its arrival time at 0500 but the CDO is busy preparing the morning brief. The assistant duty officer (ADO) checks SANS during the morning watch relief, but he only pauses long enough to glance at the list and log the time change. In the midst of watch reliefs, morning arrivals, and briefings, the ADO forgets to pass the change to the oncoming watch. Mid-morning, the new CDO reviews the previous logs and sees the arrival change. He asks the new ADO what action has been taken. The ADO checks SANS and verifies the new arrival time. The ADO calls the unit, and leaves a message with the seaman who answers the phone. He follows it up with an email to the executive officer (XO), who is away from the unit until mid-afternoon. The ADO expects the cutter to notify the VBST about the time change. Upon returning to the unit, the XO grants liberty to her hard working crew and meets with the commanding officer (CO). The work day has ended by the time she checks her email and sees the time change. She immediately talks to the CO, sends a page to her crew, and then calls the CDO to remind them about the cutter's dockside in the morning. The CDO looks up the sector's vessel status list and sees the PIKE is available. He passes that information to the oncoming watch later that evening.   After the watch turns over, the oncoming watch sends a tasking email to the cutter and the VBST. However, it's 2130, and the VBST is standing on the dark pier ready to conduct the boarding.

When you replace the over-extended, multitasked human element with an always-running service, communications improve. In the service-enabled HLS escort scenario, a monitoring service (described in the next section) continuously checks a legacy system and immediately alerts the duty officer when the target vessel changes its arrival. A tasking service automatically identifies the schedule conflict for the assigned unit and proposes alternative resources to select. Once the CDO has made a choice, a communications service automatically generates and sends messages alerting all involved units to the change in tasking, giving all concerned ample time to adjust their schedules. A service replaces communications that would otherwise require humans to perform telephone or email transmittals.

With a better understanding about service messaging in the SOA, we can address communications between the SOA and people via instant messaging. Instant messaging has become a valuable command and control communications capability. Messages are sent via computer or Short Message Service (SMS) over a cellular phone. The SOA includes services that generate instant messages, however, we must take care when creating them. A service can just as easily message one recipient as 100. Over-messaging users may glut and desensitize users, so they may miss a vital message among the fodder. Like the diverse planning and tasking needs of our various mission areas, we have diverse communications needs across different individuals in our service. Some individuals focus on details while others wish to grasp only big picture changes. Senior officers usually have less interest in minor changes than a program manager would have. The messaging services need to address these diverse needs as well, providing a common base structure applicable to all users, while allowing personalization at the individual level.

b. Legacy Communications Systems

The following legacy communications systems can provide functionality in the CGC2 SOA:

- Coast Guard Message System (CGMS): a system that transmits and receives text messages.
- Rescue 21 (R21): USCG's modernized distress communications system, providing 911-like service to mariners over VHF and UHF radio.

c. Communicating Services

The following descriptions provide the framework for creating services to meet the communicating needs described above. They were written as general communicating services we can reuse in other Coast Guard SOAs. These services create a publish and subscribe capability that will push and pull messages through the system in a manner transparent to the user:

- <u>Create Message</u>: Creates and "publishes" a message as the requestor specifies.
- <u>Receive Message</u>: Subscribes to a publishing service and typically serves as the initiating event in a work flow or business process.

- Send Message: The create message service calls this service to transmit a message. When a unit doesn't have coverage, such as a cutter or aircraft,, this service will transmit the message to the Forward Message service..
- Forward Message: Provides a store-and-forward message repository. When units do not have coverage, this service holds the message and sends it when they become available.

d. Communicating Conclusion

The messages that flow within the SOA between services and people dramatically improve the communications capabilities of the Coast Guard. Numerous proprietary and open instant messaging standards exist for us to choose from. Many government and military applications have embraced the Extensible Messaging and Presence Protocol (XMPP) open standard. The Marine Corps recently adopted XMPP as their instant messaging standard. The Coast Guard must research the available options and choose the standard that will best meet our needs. However, we should consider XMPP first.

4. Monitoring

A large part of any command and control system's success hinges on its ability to monitor the environment. The typical C2 system monitors blue (friendly) force positions, operational status, and endurance, usually in separate displays. It has sensors (e.g., radars, cameras) that monitor various environmental aspects to enhance situational awareness, but these proprietary and closed systems usually cannot share information with third parties. The CGC2 SOA monitoring services provide environmental data from isolated sensors to the SOA, allowing any participant with the appropriate permissions to access the data.

a. Discussion

Effective command and control requires monitoring, collecting and fusing a tremendous amount of information. Our situational awareness hinges on our ability to repeatedly access the appropriate information sources, evaluate the data, and make the right conclusions. We consult many data sources several times during each watch, in each operations center, in each sector, in each district, in each area. The Coast Guard expends hundreds of man-hours each day, having highly skilled people sift through disjointed information displays expecting them to correctly interpret the data and make the right judgments at the right time. However, due to the sheer information mass, we often miss that elusive data tidbit that would make it all clear. As currently practiced, fusion requires much human effort, achieves modest results, and costs a lot. The services in the monitoring functional area can automate the process a great deal, providing the most valuable information to the user.

The SARSAT system is a good example of a stovepipe sensor system. The system monitors the world's oceans for Emergency Locator Transmitters (ELT), set off when people are in distress on the sea. The U.S. Mission Control Center (USMCC) in Suitland, Maryland, monitors the entire system. When they receive an ELT, they report it to the Joint Rescue Coordination Center (JRCC) in the distress region. The JRCC then passes tasking on the Sector who tasks the unit. This entire process happens by voice telephone communications, slowing the information flow. Directly feeding this data into a service can reduce or eliminate the human activity. The new Rescue 21 system (maritime 911) is also unnecessarily stovepiped. It displays the alert position information on a computer monitor, but does not provide the data to other systems. Humans must extract and distribute Rescue 21 information.

b. Legacy Monitoring Systems

The following legacy monitoring systems can provide functionality in the CGC2 SOA:

- Automated Mutual Assistance Vessel Rescue (AMVER) System: a voluntary global reporting system to provide accurate ship positions and characteristics for vessels near a reported distress, and then divert the best-suited ship(s) to respond to that distress.
- Automated Identification System (AIS): a transponder based system onboard commercial vessels that broadcasts identification and position information.
- NLETS / NCIC: a Department of Justice database for criminal justice information including photographs and fingerprints.

- Hawkeye: a sensor-tracking system to detect, track, and identify vessel traffic.
- Lookout Lists (LOL): a federally maintained list containing individuals subject to intense scrutiny from the US government.
- Ports and Waterways Safety System (PAWSS): a surveillance and detection system using remote sensors to monitor vessels operating in U.S. ports and waterways.
- Rescue 21 Direction Finder (R21DF): a triangulation capability for VHF and UHF communications to "pin point" a radio transmission's location.
- Search and Rescue Satellite Aided Tracking system (SARSAT): a satellite system for detecting a relaying Emergency Position Indicating Radio Beacons (EPRIB) and Personal Locator Beacons (PLB) signals to the appropriate Rescue Coordination Center.
- Ship Arrival and Notification System (SANS): a database populated with Advanced Notice of Arrival information provide by ships 96 hours prior to entering U.S. territorial waters.
- Vessel Monitoring System (VMS): and AIS system for fishing vessels.
- Vessel Traffic Service (VTS): a navigation information and traffic organization system to improve situation awareness for vessels operating in certain waterways. VTS sensors include cameras, radars, and AIS.

c. Monitoring Services

The following descriptions provide the framework for creating services to meet the monitoring needs described above. The first three were written as general monitoring services for reuse in other Coast Guard SOAs.

- <u>Monitor Receive-Push</u>: Accepts data being pushed from an external source (asset, service, or system), and then forwards the data to a consumer.
- <u>Monitor Request-Pull</u>: Requests data from an external source (asset, service, or system) configured to respond to requests.
- <u>Monitor Transmitter</u>: Sends internal events to subscribers.

- Weather Forecast: Extension to the base Receive-Push service for data from the National Weather Service. Receives data forecast for a defined area.
- Environmental Limits Monitor: This coarse-grained service uses the output from Weather Forecast service and a given asset type. It uses that data to invoke the Environmental Limits Check service. This creates continuous monitoring of weather conditions and automatically alerts the user when they exceed limits.
- Rescue21DF: Extension to the base Receive-Push service for the Rescue 21 Direction Finding system. This service will receive triangulated positions from distress call for a given geographic area.
- eNOAD: Extension to the base Request-Pull service for the ship arrival and departure notification system. Initial version will get updated information for all vessels in a defined area at a defined frequency. One variation will only request information on one vessel and will be linked to a task, so that vessel arrival changes that impact CG plans will get flagged.
- AIS: Extension to the base Receive-Push service for the Automatic Identification System. This service will receive vessel position information for all vessels in a defined area.
- Blue Force Tracker: Extension to the base Receive-Push service to track USCG asset positions. This service will receive cutter, boat, and aircraft positions within a defined area.

d. Monitoring Summary

The Coast Guard employs many different stovepipe systems to monitor the maritime domain. The services in the monitoring functional area expose the functionality and data from those stovepipes, allowing the SOA to expose them for others to access and exploit.

5. Reporting

a. Discussion

The reporting functional area operates at several layers. It exchanges pertinent data with existing legacy systems. It fuses information collected by monitoring services into relevant data that can be used for planning and tasking. It provides a means for retrieving information or statistics on resources expended. Strategic planning and service growth especially benefit from this functionality. Finally, it contains internal administrative services necessary to manage and audit data entries for accuracy and process adherence.

For example, as the CDO closes a SAR case the administrative audit automatically occurs. The audit will ensure the data accuracy and correlation to all sources associated with that entry. The audit will also ensure that all assigned tasks or defined business processes were completed. It forwards exceptions to information quality or process adherence to the case owner in an exception report via the messaging services.

b. Legacy Reporting Systems

The following legacy reporting systems can provide functionality in the CGC2 SOA:

- Abstract of Operations (AOPS): a database for recording Coast Guard asset (cutter, boat, aircraft) employment information.
- Local Notice to Mariners (LNM): a notification system to alert mariners about Aids to Navigation (AtoN) discrepancies, outages, corrections, and hazards to navigation.
- Marine Information for Safety and Law Enforcement (MISLE): four integrated applications for recording information about LE, Marine Safety, SAR, and other missions.
- MHS-OPS: a prototype system to create standardized, operational planning for Homeland Security operations across the Chain of Command levels (HQ, Area, District, Sector, Unit).
- Situation Report (SITREP): a standard report generated to inform the Chain of Command about on-scene conditions and mission progress.
- Status Board: a display showing subordinate assets (cutters, boats, aircraft, teams, personnel), their conditions and current actions, typically done by hand on a dry-erase board.

c. Reporting Services

The following descriptions provide the framework for creating services to meet the reporting needs described above. These three were written as general reporting services for reuse in other Coast Guard SOAs:

- <u>Report Request</u>: Creates a request for a report containing specific information from specific sources.
- <u>Report Aggregator</u>: Gathers information from other services and sends it to the report generator.
- <u>Report Generator</u>: Creates the report in the requested format and delivers it to the requestor.
- <u>Database Query</u>: Performs SQL Data Manipulation Language Select, Insert, Update, and Delete database queries.
- <u>Case Auditor</u>: Verifies the process completion and information contained in an case (e.g., SAR, law enforcement), with detailed exception reporting.
- <u>Create Log Entry</u>: Creates an electronic log entry, can link to another specific service execution, creating an official record of Coast Guard actions.
- <u>Review Log</u>: Displays logs so that users can browse official records, hyper-links with logs allow users to review when and how services were utilized.

d. Reporting Summary

The reporting services unlock the data trapped in legacy stovepipes. In doing this, they have the potential to reach the most users and improve the information quality they utilize. The ability to extract information quickly and easily will improve many users' effectiveness. SOA's customizable nature will allow users to configure the reporting services to exploit previously low value information in new and powerful ways.

E. QUALITY ATTRIBUTES

The Software Engineering Institute defines a quality attribute as "a property of a work product or goods by which its quality will be judged by some stakeholder or stakeholders. The quality attribute requirements … have a significant influence on the software architecture of a system." ("Software

Architecture Glossary") Quality attributes are the product aspects that stakeholders deem most important to success, either by delivering what stakeholders desire or avoiding things they can't accept. We emphasize the following quality attributes for the CGC2 SOA: interoperability, security, usability, extensibility, and scalability. Other quality attributes may also apply, but these five are essential.

1. Utility Tree

Utility trees provide a top-down, structured method for generating scenarios to define quality attributes concretely. The utility tree's nodes show important quality goals and the leaves hold scenarios exemplifying those goals. We have produced a utility tree for the CGC2 SOA. Figure 8 below shows the first three levels. This tree stops at the quality attribute refinement level, before showing the specific quality attribute scenarios. Individual quality attribute descriptions later in this section carry the process through and depict detailed scenarios. We've numbered the quality attributes using a dot notation, for example numbering "Security – ensure releasability" as 2.4. The trees show these numbers in blue.

A utility tree also encourages stakeholders to prioritize the quality attribute requirements in two ways: "(1) by the importance of each scenario to the success of the system and (2) by the degree of difficulty posed by the achievement of the scenario, in the estimation of the architect." (Clements, Kazman, Klein 55) Relative rankings High (H), Medium (M) and Low (L) indicate the priorities we assigned. The scenarios marked (H, H) become the focus of architecture development effort because they represent current and future driving forces on the architecture. We indicate our priorities above each quality attribute scenario.

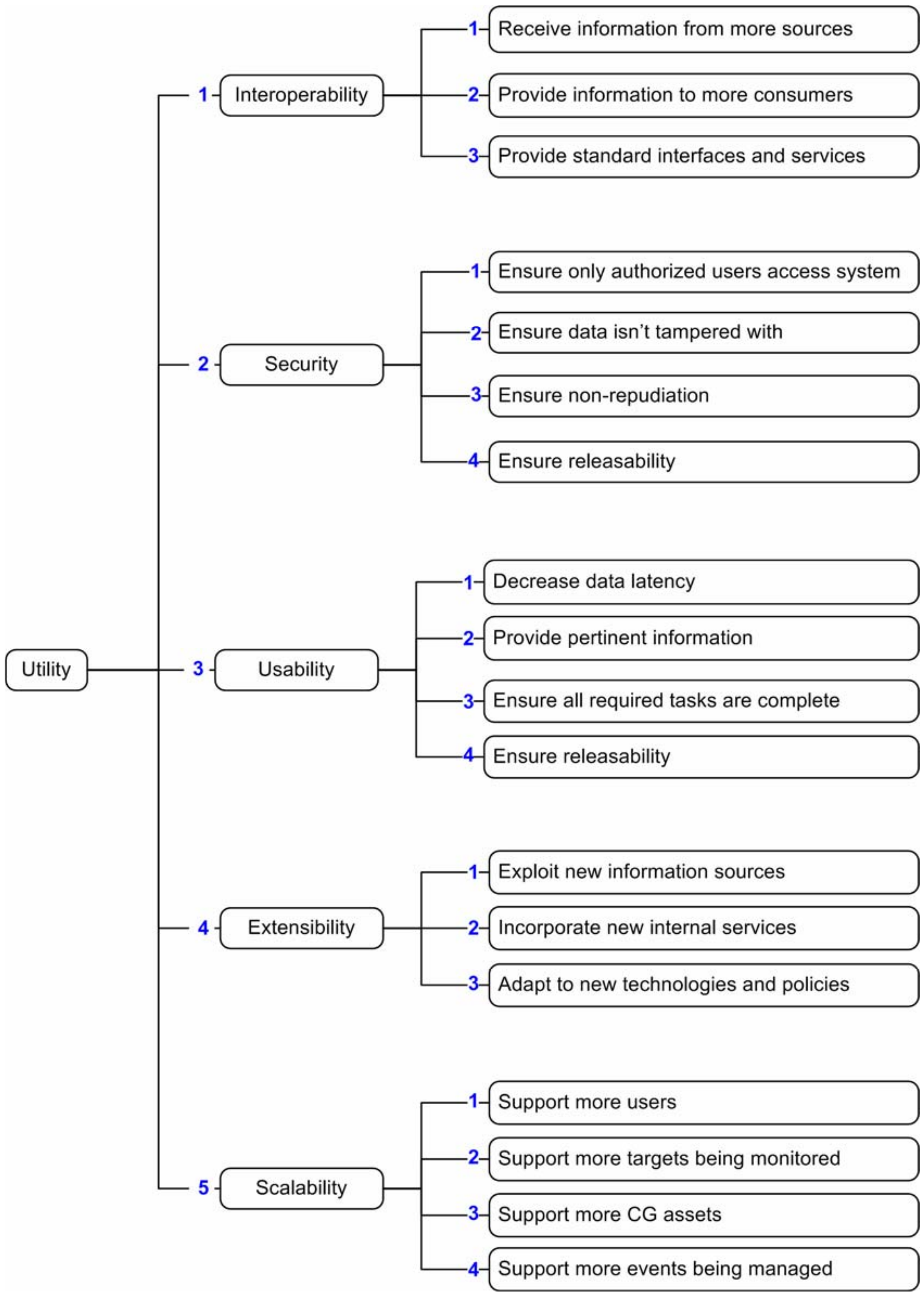
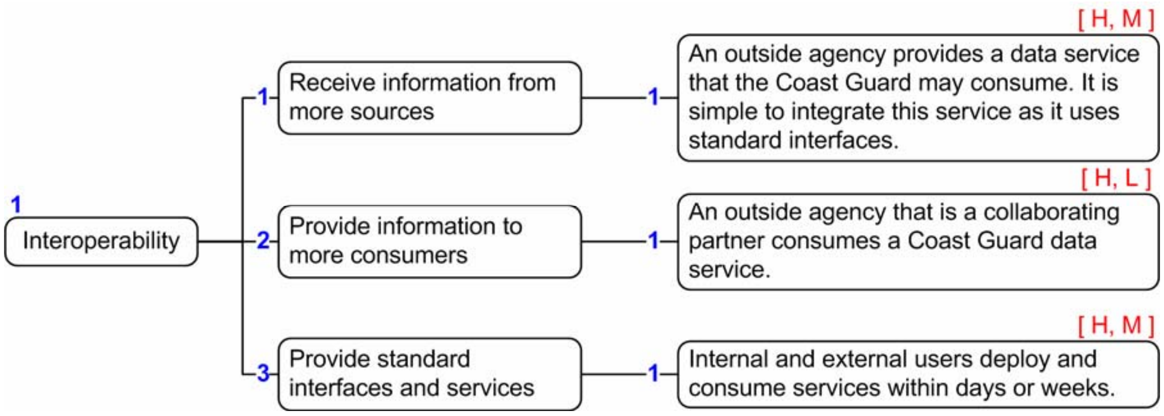

Utility
1 Interoperability
1 Receive information from more sources
2 Provide information to more consumers
3 Provide standard interfaces and services
2 Security
1 Ensure only authorized users access system
2 Ensure data isn't tampered with
3 Ensure non-repudiation
4 Ensure releasability
3 Usability
1 Decrease data latency
2 Provide pertinent information
3 Ensure all required tasks are complete
4 Ensure releasability
4 Extensibility
1 Exploit new information sources
2 Incorporate new internal services
3 Adapt to new technologies and policies
5 Scalability
1 Support more users
2 Support more targets being monitored
3 Support more CG assets
4 Support more events being managed

Figure 8. Utility Tree (top level)

2. Quality Attribute – Interoperability (1.0.0)

Interoperability assures the communicating entities' can share specific information and operate on it according to an agreed-upon operational semantics. (O'Brien, Bass, Merson 4) The scenarios on the right side of Figure 9 illustrate our SOA's unique interoperability requirements.



1
Interoperability
1
Receive information from more sources
1
[H, M]
An outside agency provides a data service that the Coast Guard may consume. It is simple to integrate this service as it uses standard interfaces.
2
Provide information to more consumers
1
[H, L]
An outside agency that is a collaborating partner consumes a Coast Guard data service.
3
Provide standard interfaces and services
1
[H, M]
Internal and external users deploy and consume services within days or weeks.

Figure 9. Utility Tree – Interoperability

3. Quality Attribute – Security (2.0.0)

Security has many different aspects, but generally exists when users, applications, and services can only perform authorized actions. The following four principles are broadly used to define computer security:

- Confidentiality – only authorized subjects can access the information or service.
- Authenticity – verification that the indicated author/sender is the one responsible for the information.
- Integrity – information is not corrupted.
- Non-repudiation – a message or action cannot later be denied by any participant.

(O'Brien, Bass, Merson 12)

The scenarios on the right side of Figure 10 illustrate our SOA's unique security requirements.

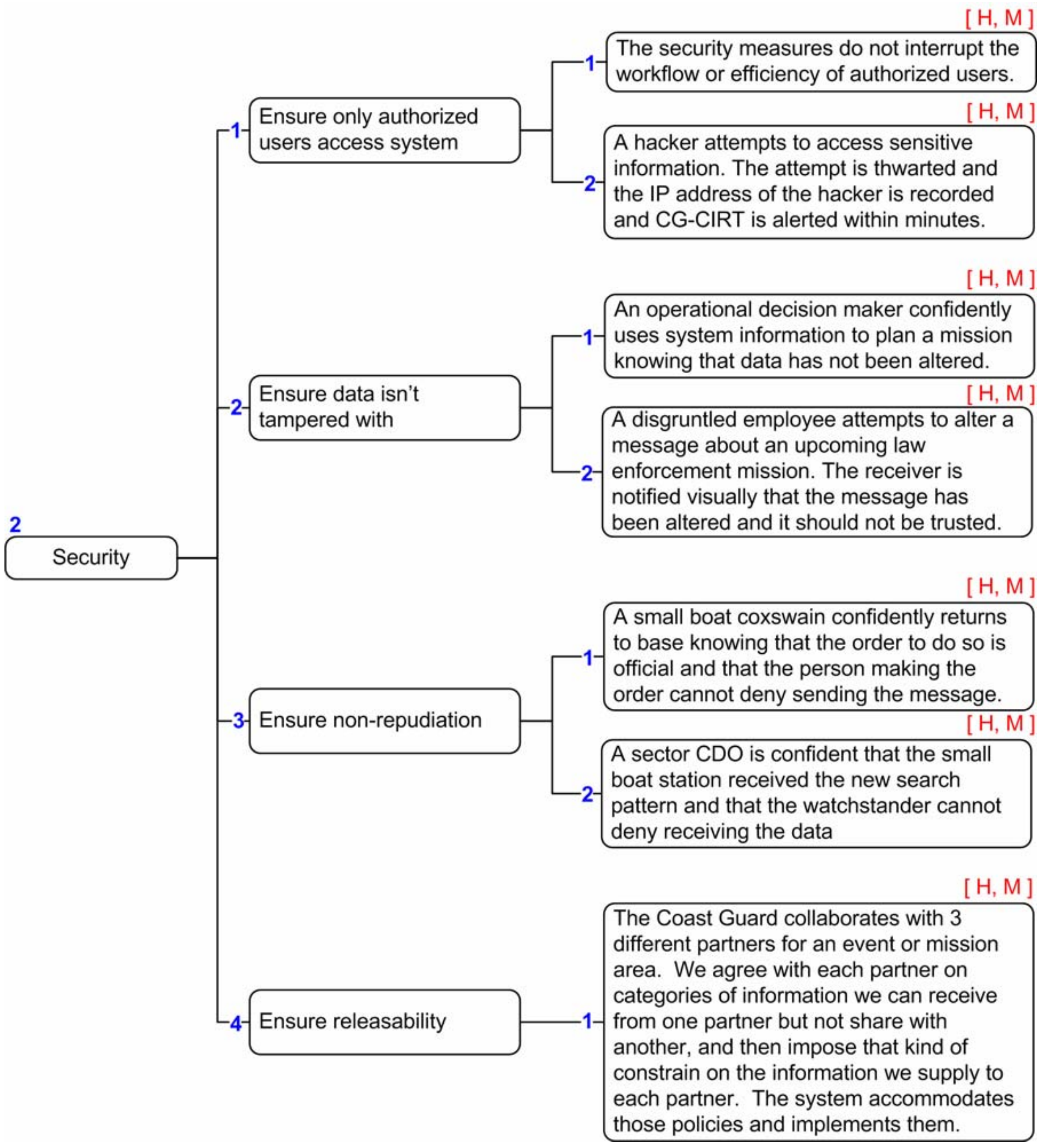

2
Security
1 Ensure only authorized users access system
1 [H, M] The security measures do not interrupt the workflow or efficiency of authorized users.
2 [H, M] A hacker attempts to access sensitive information. The attempt is thwarted and the IP address of the hacker is recorded and CG-CIRT is alerted within minutes.
2 Ensure data isn't tampered with
1 [H, M] An operational decision maker confidently uses system information to plan a mission knowing that data has not been altered.
2 [H, M] A disgruntled employee attempts to alter a message about an upcoming law enforcement mission. The receiver is notified visually that the message has been altered and it should not be trusted.
3 Ensure non-repudiation
1 [H, M] A small boat coxswain confidently returns to base knowing that the order to do so is official and that the person making the order cannot deny sending the message.
2 [H, M] A sector CDO is confident that the small boat station received the new search pattern and that the watchstander cannot deny receiving the data
4 Ensure releasability
1 [H, M] The Coast Guard collaborates with 3 different partners for an event or mission area. We agree with each partner on categories of information we can receive from one partner but not share with another, and then impose that kind of constrain on the information we supply to each partner. The system accommodates those policies and implements them.

Figure 10. Utility Tree – Security

4. Quality Attribute – Usability (3.0.0)

Usability measures the quality of a user's experience while interacting with information or services. (O'Brien, Bass, Merson 11) A system that does what the user wants, when they want it done has high usability. The scenarios on the right side of Figure 11 illustrate our SOA's unique usability requirements.

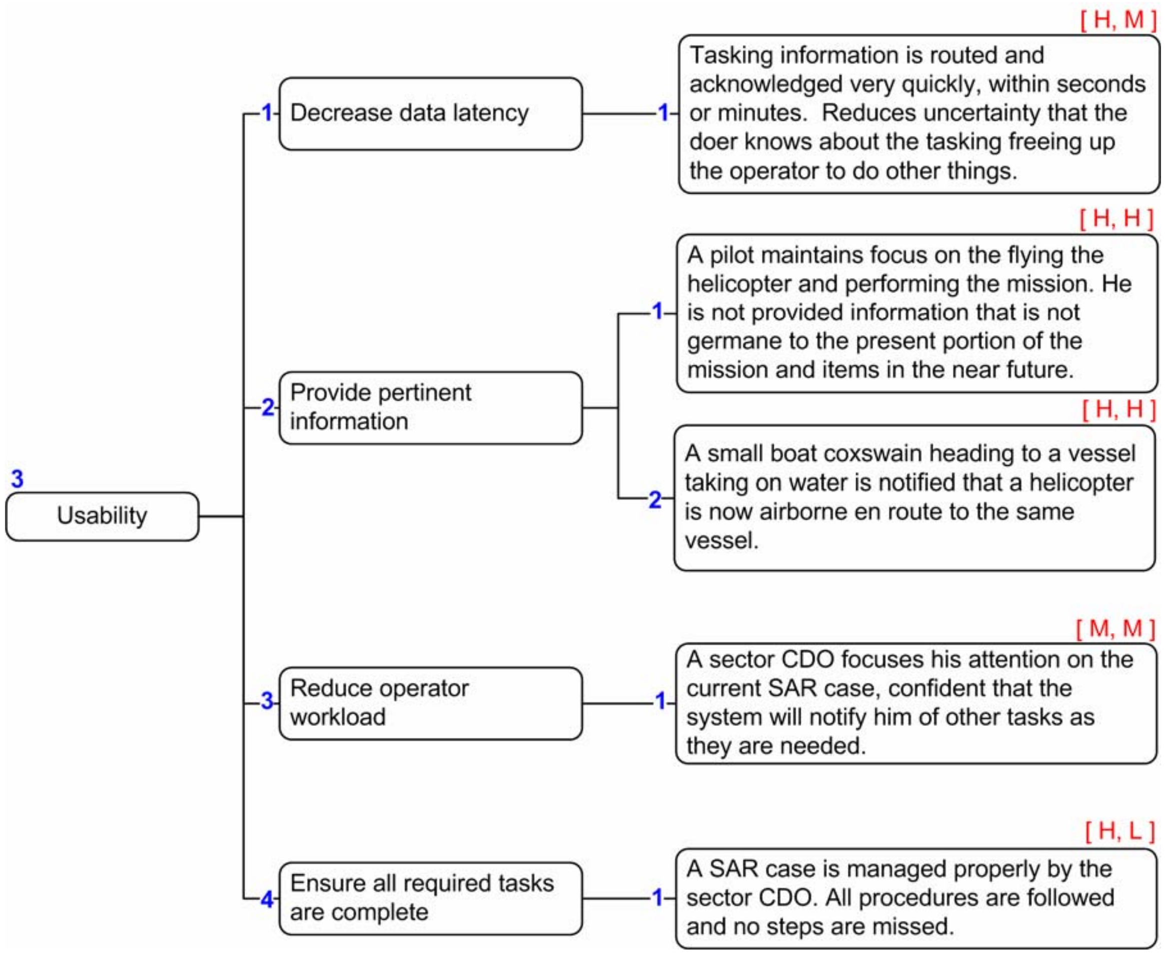

[H, M]
[H, H]
[H, H]
[M, M]
[H, L]
3
Usability
1 Decrease data latency
2 Provide pertinent information
3 Reduce operator workload
4 Ensure all required tasks are complete
1 Tasking information is routed and acknowledged very quickly, within seconds or minutes. Reduces uncertainty that the doer knows about the tasking freeing up the operator to do other things.
1 A pilot maintains focus on the flying the helicopter and performing the mission. He is not provided information that is not germane to the present portion of the mission and items in the near future.
2 A small boat coxswain heading to a vessel taking on water is notified that a helicopter is now airborne en route to the same vessel.
1 A sector CDO focuses his attention on the current SAR case, confident that the system will notify him of other tasks as they are needed.
1 A SAR case is managed properly by the sector CDO. All procedures are followed and no steps are missed.

Figure 11. Utility Tree – Usability

5. Quality Attribute – Extensibility (4.0.0)

Extensibility provides the ability to add new features or components to the existing services without affecting other services or parts of the system. (O'Brien, Bass, Merson 17) Extensibility requires the architecture to consider future growth and adapt to a changing environment. The scenarios on the right side of Figure 12 illustrate our SOA's extensibility requirements.

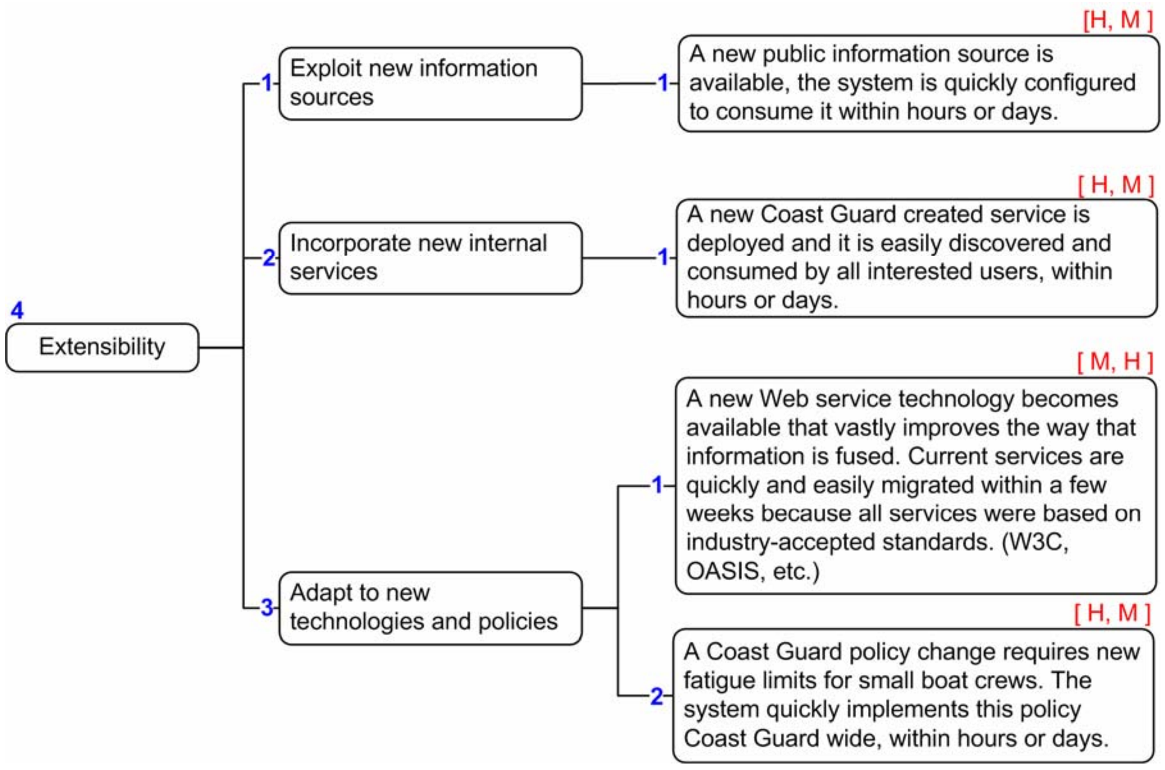

4
Extensibility
1
Exploit new information sources
1
[H, M]
A new public information source is available, the system is quickly configured to consume it within hours or days.
2
Incorporate new internal services
1
[H, M]
A new Coast Guard created service is deployed and it is easily discovered and consumed by all interested users, within hours or days.
3
Adapt to new technologies and policies
1
[M, H]
A new Web service technology becomes available that vastly improves the way that information is fused. Current services are quickly and easily migrated within a few weeks because all services were based on industry-accepted standards. (W3C, OASIS, etc.)
2
[H, M]
A Coast Guard policy change requires new fatigue limits for small boat crews. The system quickly implements this policy Coast Guard wide, within hours or days.

Figure 12. Utility Tree – Extensibility

6. Quality Attribute – Scalability (5.0.0)

Scalability provides the ability to function well (without degradation of other quality attributes) when the system increases in size or volume in order to meet users' needs. (O'Brien, Bass, Merson 16) Designing for scalability requires understanding the bottlenecks in the system and then applying a horizontal (distributing work to other machines) or vertical (upgrade to more powerful machine) solution. The scenarios on the right side of Figure 13 illustrate our SOA's unique scalability requirements.

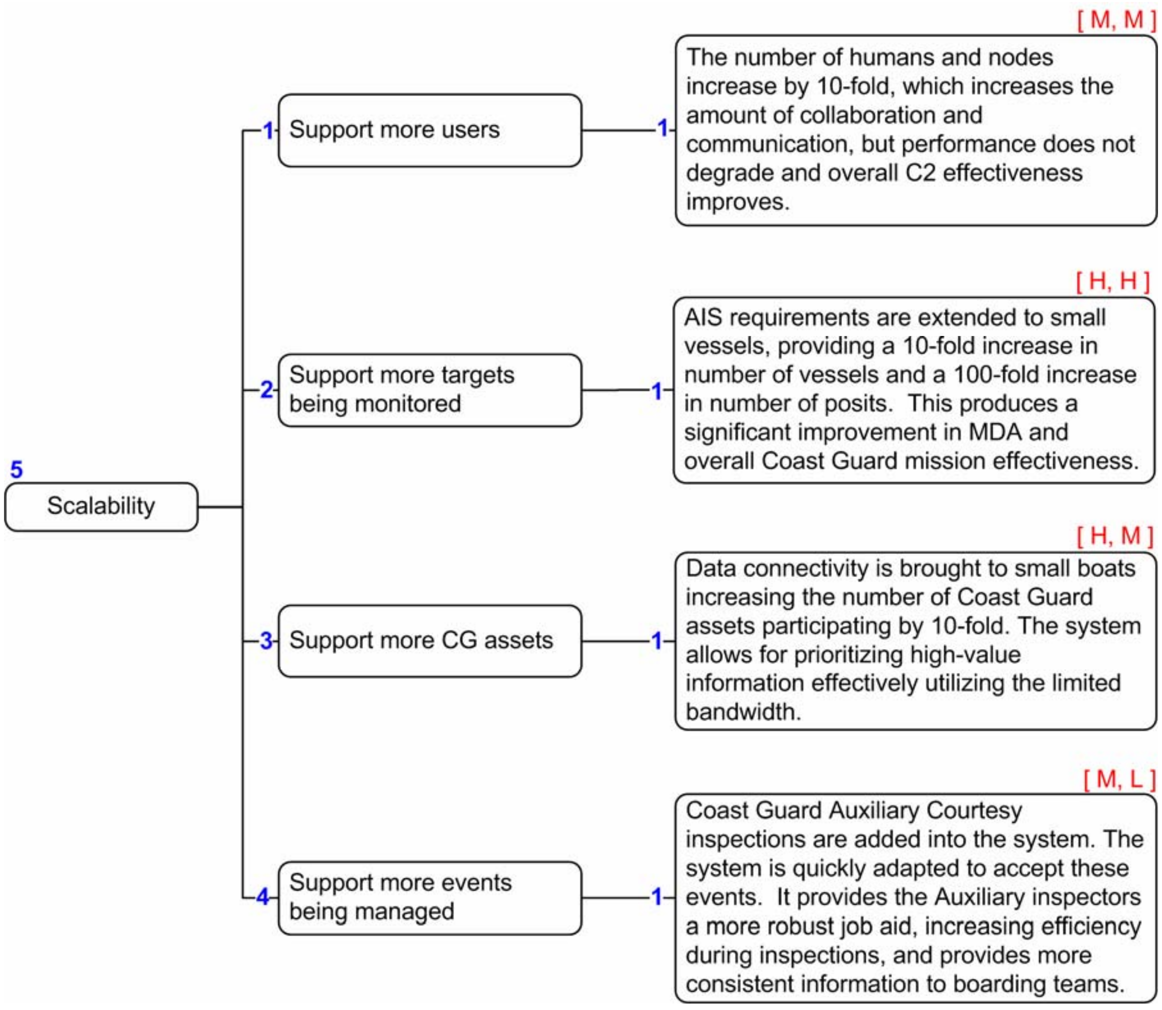

5
Scalability
1 Support more users
1
[M, M]
The number of humans and nodes increase by 10-fold, which increases the amount of collaboration and communication, but performance does not degrade and overall C2 effectiveness improves.
2 Support more targets being monitored
1
[H, H]
AIS requirements are extended to small vessels, providing a 10-fold increase in number of vessels and a 100-fold increase in number of posits. This produces a significant improvement in MDA and overall Coast Guard mission effectiveness.
3 Support more CG assets
1
[H, M]
Data connectivity is brought to small boats increasing the number of Coast Guard assets participating by 10-fold. The system allows for prioritizing high-value information effectively utilizing the limited bandwidth.
4 Support more events being managed
1
[M, L]
Coast Guard Auxiliary Courtesy inspections are added into the system. The system is quickly adapted to accept these events. It provides the Auxiliary inspectors a more robust job aid, increasing efficiency during inspections, and provides more consistent information to boarding teams.

Figure 13. Utility Tree – Scalability

7. Operational Examples

Table 2 below contains nine vignettes that demonstrate the CGC2 SOA's desired aspects. The right column links each vignette to the applicable scenarios from the utility trees above using the dot notation. This table shows the relationships between Coast Guard missions and the architecture's quality attributes.

Vignettes	QA Scenario
Marine Safety	
[Search & Rescue] A cruise ship carrying 1000 passengers catches fire 100 miles off the coast of North Carolina. Aircraft and cutters from two Coast Guard districts respond. Ten level 1 and level 2 trauma centers are contacted. AMVER identifies 10 vessels in the vicinity, they are contacted and assist.	1.2.1 3.1.1 3.3.1 3.4.1
[Boating Safety] The Coast Guard implements a Safe Boating portal for the public. It provides weather forecasts, notice to mariners information, and the ability to create and file a Float Plan. This web site does not break under heavy seasonal load (e.g., summer holiday weekends). The information entered is easily shared with local units and emergency response agencies.	1.1.2 3.4.1 4.2.1 5.1.1 5.4.1
National Defense	
[Homeland Security] Intel from a new data source detects a potential threat on a cargo container bound for the U.S. The vessel is boarded off shore and the container is found carrying hundreds of illegal weapons. This event involves the Coast Guard, Customs and local port authority.	1.1.1 1.1.2 2.1.1 4.1.1
Maritime Security	
[Drug Enforcement] A WMEC on a regularly scheduled LE patrol in the Caribbean Sea boards a foreign flagged high interest vessel and discovers 3000 pounds of cocaine. The U.S. State Department, U.S. Department of Justice, and the foreign government are also involved.	1.1.1 2.1.1 2.2.1 2.3.2
[LMR] A WHEC reports dozens of Russian vessels illegally fishing in the "Donut Hole" in Alaska. The WHEC CO reports that he met with strong resistance while attempting to board one of the vessels and he is asking for support.	2.2.1 2.3.1 3.1.1 3.2.1
[LMR] A short duration seasonal fishery opens requiring increased law enforcement effort and SAR response readiness. This day or week long event involves the National Marine Fisheries Service, state Fish and Wildlife agencies, and National Oceanographic and Atmospheric Agency attorneys.	1.1.2 1.2.1 5.1.1 5.2.1
Maritime Mobility	
[ATON] A hurricane off the east coast forces over 100 buoys off station and damages hundreds more fixed aids. ATON assets from multiple districts respond to survey the waterways and reposition the aids.	1.2.1 3.1.1 5.2.1 5.3.1
[VTS] A new Vessel Traffic Service is established in a large commercial port. This new unit will require 50 new users, track over 100 vessels a day, and will exchange information with two port authorities, the pilots, and 50 companies.	5.1.1 5.2.1 5.4.1
Protection of Natural Resources	
[Oil Spill] A super tanker runs aground in the Straits of Juan De Fuca spilling millions of gallons of crude oil, jeopardizing hundreds of miles of U.S. and Canadian coastline. The response effort includes multiple U.S. and Canadian Coast Guard assets, as well as federal and local government agencies.	1.1.1 1.1.2 3.3.1 4.1.1

Table 2. Operational Examples and Corresponding Quality Attributes

F. CONCEPTUAL DATA MODEL (CDM)

This section outlines the CGC2 SOA data model. The actual CDM process will be “a serious data modeling exercise that typically requires the input of highly experienced analysts and architects. The end result is a set of custom standards for the enterprise.” (Gabriel) As such, the conceptual diagrams presented here show only a small portion of the full model required to implement a functional system. We chose to focus on the Planning and Tasking functional areas because they provide examples most readers will easily understand. At this point it’s important to clarify our terminology so we don’t confuse the terms planning and tasking. In this chapter’s “Functional Areas” section, planning and tasking are verbs, or actions the services perform. In this section, planning and tasking are nouns, or concepts represented by the data models shown. Although we present incomplete data models, they provide concrete data organization examples within the CGC2 SOA system. The figures below show XML schema diagrams. The rectangles represent individual CDM elements (e.g., Asset), but they do not contain specific data from the example (e.g., USCGC RUSH, a high endurance Coast Guard cutter).

1. Planning Element

A plan includes elements for the commander’s intent, the assets employed, the operating area, the plan type, and the target objects. The PlanType element contains related missions, tasks, and other plans. This data model works for a strategic plan and its supporting operational plans. Figure 14 shows a conceptual view of the Plan element.

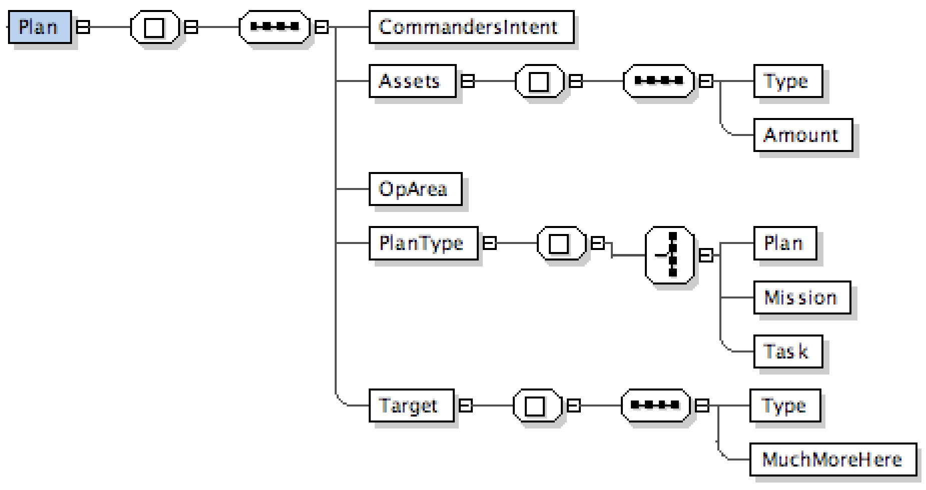


Figure 14. Data Model – Planning Element

To illustrate the data model, consider the following example. District 14 creates a fisheries enforcement operations plan. In addition to the typical operations plan information (why, who, how, what, when, where) it includes USCGC RUSH's and Air Station Barbers Point's missions and tasks. In this conceptual data model, a task can occur as a Plan element or as a Mission element contained in a plan. In our examples, all tasks are Missions elements. We have no Plan-level tasks. Missions and tasks will be exemplified in the following sections.

2. Mission Element

When an asset carries out a mission, it performs multiple tasks. The CDM Mission element's structure represents this by grouping tasks performed to support the mission. Figure 15 shows a conceptual view of the Mission element. While this conceptual view lacks detail, additional elements will be added to include appropriate amplifying details when the Coast Guard creates the actual CDM.

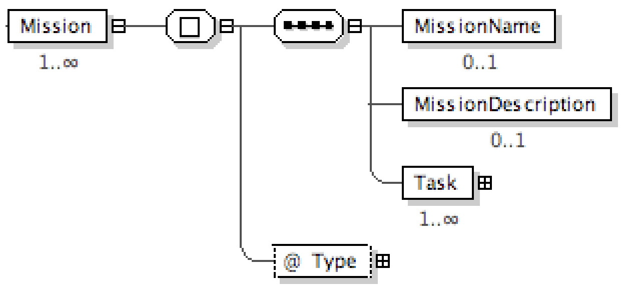


Figure 15. Data Model – Mission Element

In our example, Air Station Barbers Point's mission contains tasks for individual HC-130 surveillance flights. USCGC RUSH's mission includes patrolling a large area, employing its sensors and embarked HH-65 helicopter to locate fishing vessels. When an asset locates a fishing vessels, USCGC RUSH will intercept them and deploy its small boat, which will transport the boarding team. The team will board the vessels and enforce all applicable U.S. laws, regulations, and treaties. All four assets (e.g., RUSH, HH-65, small boat, boarding team) carry out distinct tasks related to the cutter's mission. The District 14 operations plan contains both the Air Station's and cutter's missions.

3. Task Element

The Task element's structure provides the what, when, where and other relevant details about a task. Figure 16 shows a conceptual view of the Task element.

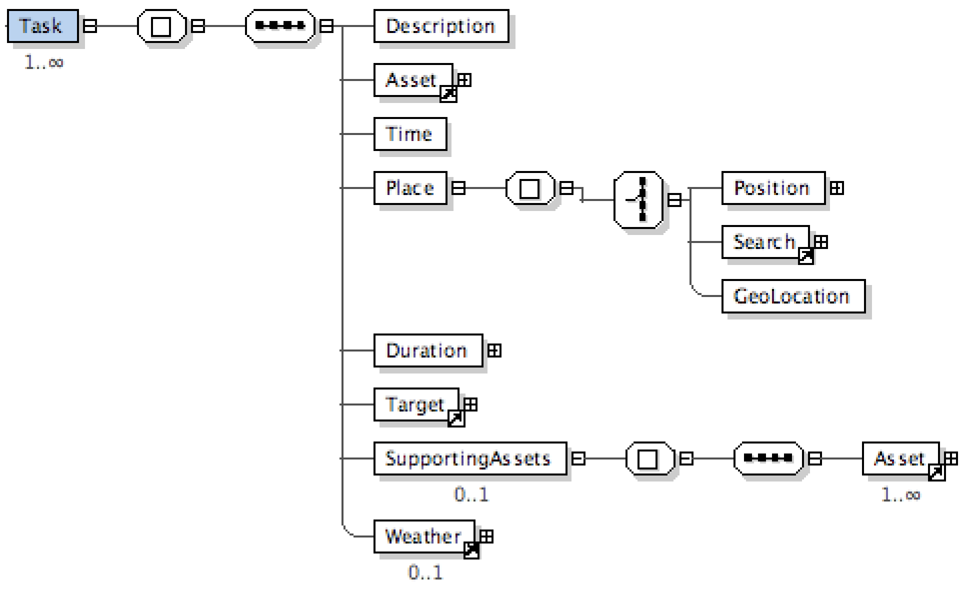


Figure 16. Data Model – Task Element

Continuing our example, we create a task for each HC-130 surveillance flight, each HH-65 flight, and multiple tasks for USCGC RUSH's patrol. The target in the Task element can be either a general target type (e.g., fishing vessels) or a specific object (e.g., F/V BIG KAHUNA) represented by the Target element shown in Section F.5. below. During a HH-65 task the helicopter locates F/V BIG KAHUNA 15 miles from USCGC RUSH. USCGC RUSH generates a task to intercept the vessel, a task for the small boat, and a task for the boarding team.

4. Asset Element

Assets include the aircraft, cutters, boats, vehicles, teams, and individuals that perform Coast Guard missions. The Asset element models an asset's capabilities (e.g., speed, range) and limitations (e.g., weather, endurance). Figure 17 shows a conceptual view of the Asset element. Assets in our example include the USCGC RUSH, HH-65, small boats, boarding teams, and HC-130. Note that the data element below can easily incorporate vehicles and vessels from local police, fire departments, and other emergency response organizations.

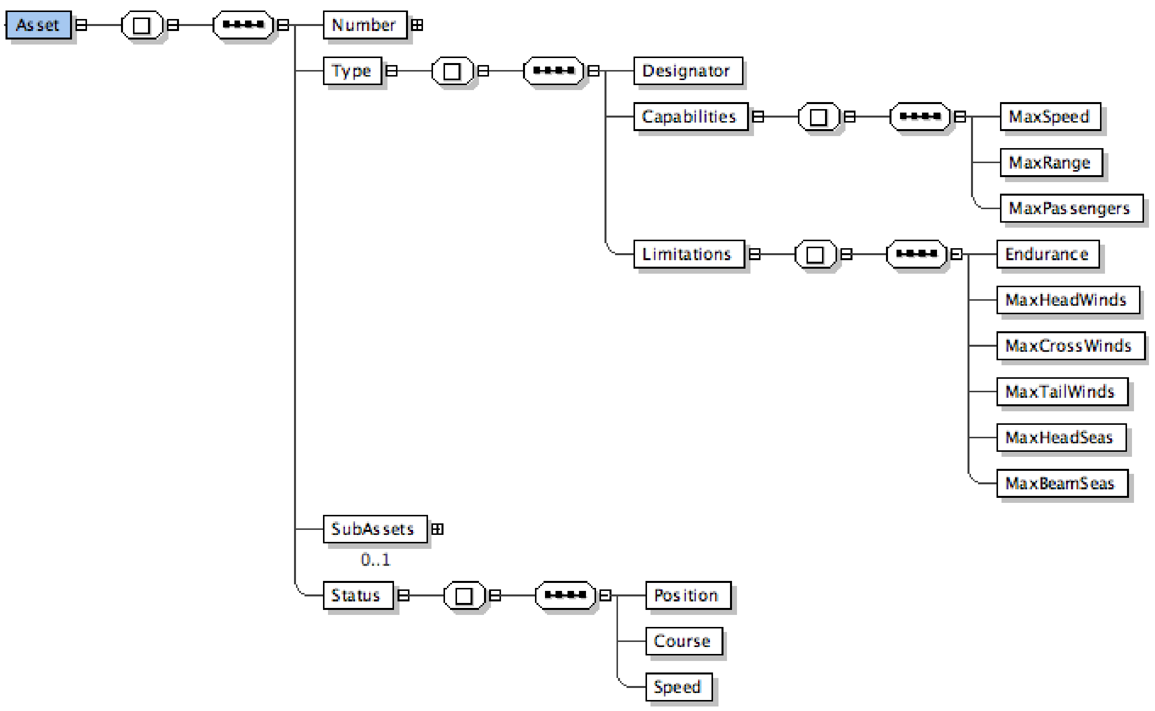


Figure 17. Data Model – Asset Element

5. Target Element

Targets are things we want to track or find. Targets exist at two levels in the CDM. The Plan and Task elements utilize general target types (e.g., fishing vessels). Tasks can also contain information about a specific object (e.g., F/V BIG KAHUNA) with details that enable an asset to track or find it. The CDM's Target element contains the needed details. Figure 18 shows a conceptual view of the Target element.

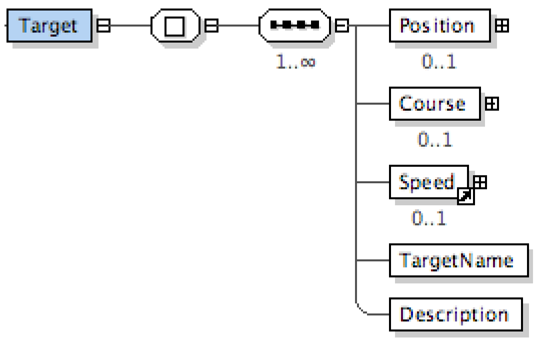


Figure 18. Data Model – Target Element

6. CDM Conclusion

A complete CGC2 SOA data model capturing all five functional areas obviously requires much more detail and many more elements. As stated earlier, creating a CDM requires experienced and skilled analysts and architects. A complete CDM exceeds this thesis' scope. We have introduced a basic framework that supports our command and control functional areas. The Coast Guard has already begun modeling data elements in existing systems with the Enterprise Data Catalogue project. While this effort will not produce a CDM, it moves us in the right direction and will provide supporting documentation to create a sound command and control CDM when the time comes.

7. Information Exchange Models

The previous sections have described an information sharing data model within the Coast Guard. However, we also need to share information with multiple federal, state, local, and foreign government agencies. That information sharing requires a different data model. Communities of Interest (COI) are collaborative groups that create an accepted information exchange vocabulary relating their shared goals, interests, and objectives. COI data models are often called information exchange models. Two notable command and control information exchange models include:

- Joint Consultation Command & Control Information Exchange Data Model (JC3IEDM) – decade-long NATO endeavor to create a command and control information exchange model.
- National Information Exchange Model (NIEM) – U.S. federal government project to create "enterprise-wide information exchange standards and processes that can enable jurisdictions to effectively share critical information in emergency situations, as well as support the day-to-day operations of agencies throughout the nation." (*www.niem.gov*)

COI data models rarely function as the CDM within any one organization, since they exist to exchange information between community members. They are not built to meet any one member's individual needs. This is true of JC3IEDM and NIEM. Neither model meets the Coast Guard's needs for internal use, because they don't meet our unique command and control needs. However, we must consider widely accepted COI models like JC3IEDM and NIEM when creating our CDM. The ability to quickly and accurately translate information between them and our CDM will provide unprecedented data sharing with other agencies.

8. Maritime Information Exchange Model (MIEM)

A COI developed data model occasionally does meet an organization's content, scope, and complexity needs. When that happens, the data model can be utilized within the organization's larger CDM. The Navy's Comprehensive Maritime Awareness Joint Capability Technology Demonstration (CMA JCTD) produced one such model. The Maritime Information Exchange Model logically groups data by epochs in the vessel's history. This links sensor data for vessel positions and all available data about cargo, people, companies, and facilities associated with the vessel. Should the MIEM become the standard for Maritime Domain Awareness data sharing, it could easily replace the Target element in our CDM's Task. This would enable the CGC2 SOA to accept a Maritime Object from an intelligence fusion system and pass it on to an asset in a Task. A CGC2 SOA Case that contained both the Coast Guard boarding results and the related Maritime Object would provide solid evidence for our law enforcement action. The Coast Guard could forward it on to the Department of Homeland Security or Department of Justice for criminal prosecution. Because those agencies IT systems can accept MIEM formatted data, the attorneys can automatically utilize the case file. This semantic interoperability perfectly demonstrates the powerful combination of SOA and shared data models. Figure 19 shows the MIEM's base element, the MaritimeObject.

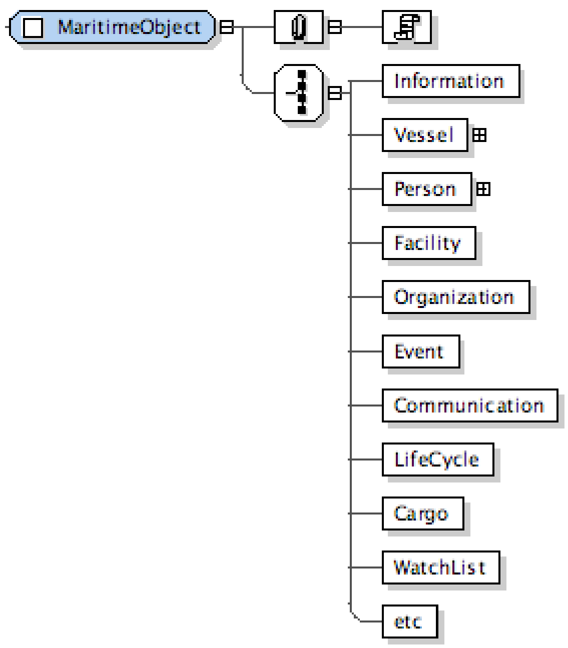


Figure 19. MIEM – Maritime Object

G. PRODUCT LINE ARCHITECTURE FOR COMPOSITE APPLICATIONS

1. Composite Applications

Composite applications interact with users by providing the necessary user-level services to create an SOA user interface. We compose applications by coupling several different services, data stores, and user interfaces using standardized message layers. Loosely coupled frameworks allow individual nodes in a distributed system to change without affecting or requiring change in any other part of the system. The composite application's components can be mixed and matched, like Lego blocks, allowing developers to create many different applications with relatively few services. Figure 20 shows a composite application for command and control with a customization layer to provide each user with the functionality and presentation he or she wants.

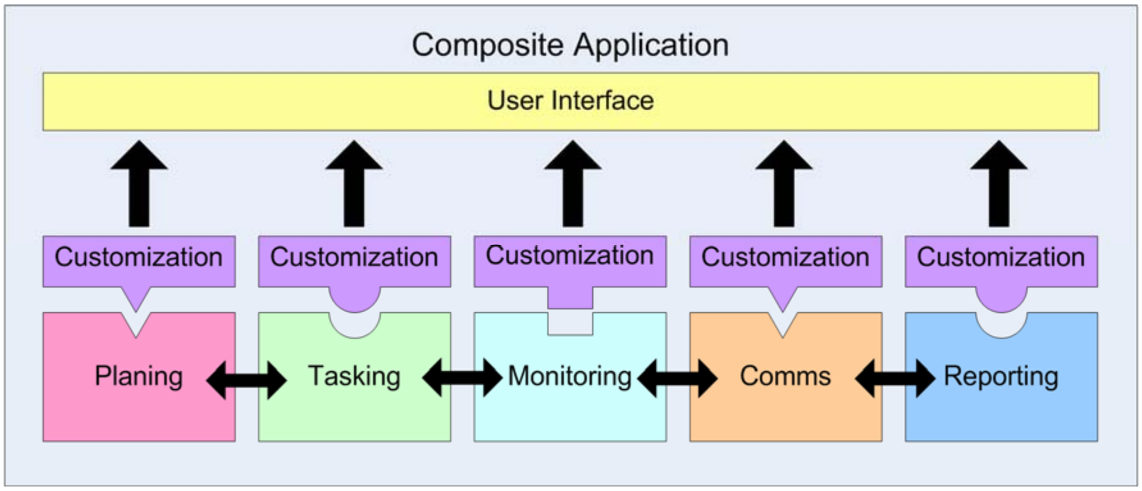


Figure 20. Example Composite Application

2. Product Line Architectures

As the Coast Guard follows the Commandant's mandate to shift our information technology infrastructure to SOA, we will make many different composite application variations. The Coast Guard should obviously develop them with an adaptive and efficient (faster, cheaper, more reuse) method. Product Line Architectures provide such a method. A PLA helps developers implement a family of related software products that address a variety of similar application requirements by composing generic reusable components. The PLA defines how the components function and interact to create the required product. The Software Engineering Institute defines a PLA-based family of products this way:

> A set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of reusable core assets in a prescribed way. ("Software Architecture Glossary")

Most successful PLAs generalize and evolve from successful products, and therefore the Coast Guard won't be in a position to create a credible PLA until after some composite applications are built. Once the first few composite applications exist, we can identify the necessary architecture and understand how to generalize specific implementations into reusable frameworks and components. We can then begin to consider how many valuable components we have, where we find them, how much to invest in them, and where to start. We need to think about our product line as a portfolio of assets, with each service or component evaluated based on its individual performance and importance to the product line as a whole. We want to work intelligently by focusing on areas of greatest value first, to discover and exploit the PLA early. The goal is to deliver big value and reap big rewards by reapplying lessons learned and reusing components extracted from previous applications and PLA endeavors. We will focus on concentrating in depth on one area, iterating to develop one PLA and systematically reuse components. Figure 21 shows a PLA for command and control, with services as the reusable components.

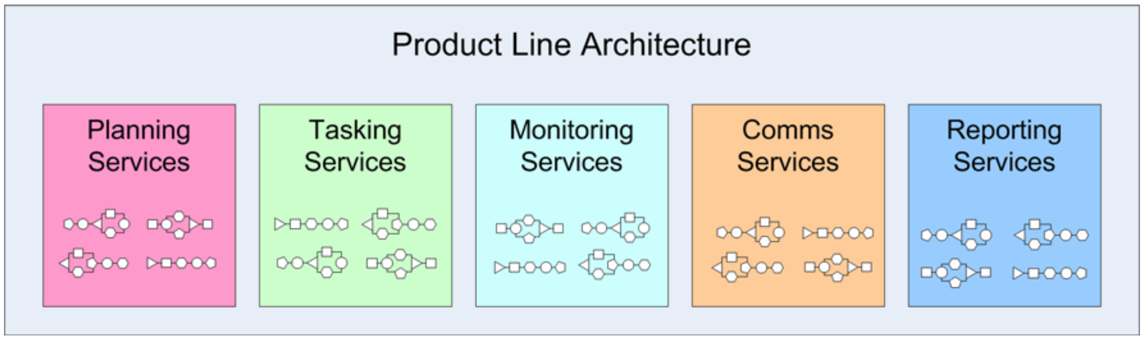


Figure 21. Command and Control SOA as PLA

3. Mashability

The purple "customization" adaptors in Figure 20 represents many different options for tailoring each component. While many issues arise in creating customizable components, we find one aspect critical for each component. We call this aspect *mashability*[1]. Each component must be *mashable* in at least three dimensions; human-computer interface, world model, and C2 functions.

[1] The term *mashup* describes a web page or application that combines data from two or more external sources. We use the term *mashability* to describe a component's ability to be mashed, or combined, with other components.

- Human-Computer Interface: A human user ultimately receives information from a component. This dimension determines the methods for physical display characteristics (size, position) and the different options for combining its output with that of other components. Most readers familiar with mashups only think of this aspect of mashability. A nautical chart from one source, weather from another component, and AIS data from a third mash easily on a graphical geographic display. These three merge to present the user with new and meaningful information.
- World Model: A "world model" represents an organization's situation awareness. It represents our best understanding of what's going on, where, why, and how. This model is the basis for *efficient thought*, a more extensive version of the OODA loop (Hayes-Roth *Hyper-beings 59*). It maintains the environment's state in three time regions; past, present, and future. It also maintains data at different levels of abstraction and aggregation, as needed for decision-making by officers responsible for vastly different geographic and temporal scopes. So each component operates on data from some select portion of the organization's world model. Each component must describe how it mashes its portion of the world model with those being used by the other components in the same application. For example the previously described GUI with weather, chart, and AIS data might have a slide bar that represents time. As the user drags the slider forward the AIS tracks and weather data step forward through time, reflecting each component's state at each future time point. The components' world model mashability makes merging components' beliefs possible.
- C2 Functionality: Each component performs one or more functions of the Efficient Thought superior decision making loop. These functions access and modify a user's world model. The components' functionalities must also be mashable. For example a component that performs some "assess situation" function must easily mash with other components that produce sensor data "observations." In this way, composite applications can assemble process chains from individual components and their outputs.

4. Conclusion

The Coast Guard will need to improve continuously its Product Line Architecture based on each iteration's successes or failures. Ultimately, we will develop services in other domains as well, and then we will want to create composite applications for human resources, logistics, and financial management, to name a few. We'll learn lessons about the architecture and its components as we build applications. Our race into the future depends on our ability to learn quickly and exploit those lessons effectively. The next chapter describes our implementation plan. It proposes a two-loop method for continuous, incremental improvement.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION PLAN

A. INTRODUCTION

This chapter provides the response to our second thesis question, "What is the optimal implementation plan for this CGC2 SOA?" Unfortunately, a typical government approach to design an architecture and develop systems that meet our command and control needs will likely fail, as do most large-scale information technology projects. These "big bang" projects have decades-long timelines and usually fail because they base their architecture on requirements collected once, during a prolonged process at the project's inception. We disagree with the "big bang" approach. We expect that our needs will change over time, and technology will continue its dramatic advance. In order to accommodate our evolving needs and capitalize on the latest technology, we think that the Coast Guard should avoid creating the CGC2 SOA as a "big bang" project.

In addition to "big bang" projects, two other common approaches are used to implement systems. Figure 22 shows their theoretical ability to provide capability over time. We call the first alternative "Build it Now" because it skims through the requirements collection and architecture definition activities and almost immediately begins building things. We call the second alternative "Incremental Evolutionary" because it aspires to deliver value while defining the architecture in response to ever changing needs. Section B describes this method in greater detail.

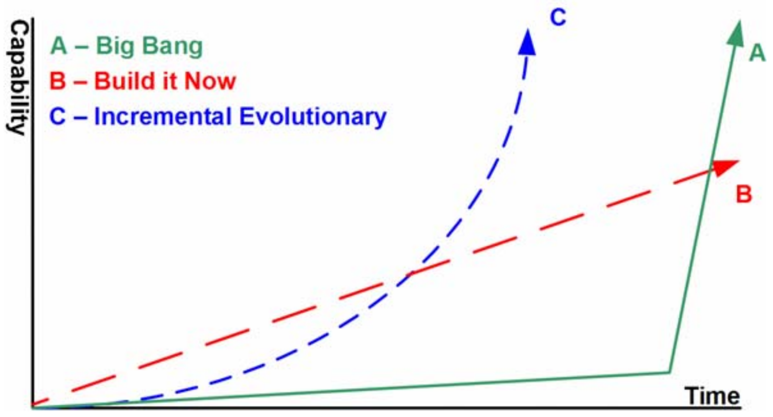


Figure 22. Theoretical Capability Derived Over Time

We recommend the “Incremental Evolutionary” approach as the best method to implement the CGC2 SOA. It recognizes the constantly changing nature of both the problem and required solution and then evolves accordingly. It also supports horizontal integration, across Coast Guard mission areas, and resists creating vertical stovepipes. Both the “Big Bang” and “Build it Now” methods fail to deliver capability as theorized. The “Big Bang” method expends precious time and resources defining a problem and potential solution, ignoring the fact that both the problem and technology available to solve it are constantly changing. The “Build it Now” approach fails because the developers fail to properly consider the long-term and widespread impacts their early decisions have on the eventual system.

Figure 23 contrasts the theoretical achievements of these various approaches with the results they usually attain in actuality. The “Big Bang” systems usually fail, thus delivering no value. The “Build It Now” approaches achieve diminishing returns over time and eventually require a start-over. An “Incremental Evolutionary” approach on the other hand, continually plans for, adapts to, and exploits predictable advances in technology to deliver more value and what Kurzweil calls “accelerating returns.” (Kurzweil 31-35)

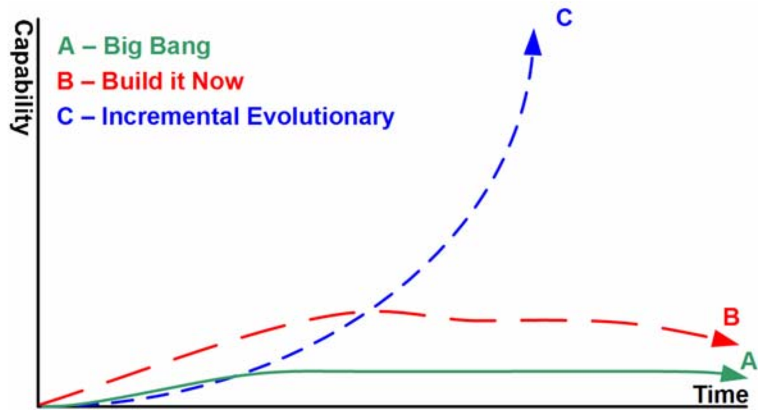


Figure 23. Actual Capability Derived Over Time

Therefore, we must adopt a flexible, rapid, and incremental implementation process that delivers some immediate value to users. To keep pace with our changing needs and advancing technology, the development cycles must produce software services or applications every six to twelve months. We must also utilize a process that will evolve the architecture as we gain experience with the service oriented methodology. We describe our incremental implementation method in Section B, followed by a proposed organizational alignment in Section C. Section D summarizes several SOA “best practices” and “worst practices” from the information technology industry and Section E addresses the impacts SOA has on quality attributes. The answer to our second thesis question provides a sound approach that outlines key activities required to create the CGC2 SOA successfully.

B. DASH-CREIGH IDeA METHOD

1. Introduction

We designed the Incremental Development Approach (IDeA) to improve the Coast Guard’s ability to implement the SOA successfully. Our method is based on agile software development practices that minimize risk by producing software in short iterations with clearly defined scope. IDeA focuses on continuous improvement of the architecture, software components, and the implementation process itself. IDeA comprises two connected loops, the Architecture Loop and the Service Development Loop (SDL). The Architecture Loop designs, evaluates, and evolves the SOA at the same time that the components (services) are created, deployed and assessed. The SDL produces and improves the actual components. We propose this approach to implement the Command and Control SOA described in Chapter III, but we purposely made it general enough for any SOA implementation.

2. Architecture Loop

The Architecture Loop begins with the vision, technical strategies, and concepts that influence the architecture. Each stakeholder brings his or her own ideas about the architecture’s design and functions. The architects and implementers need to understand SOA’s strengths and weakness when designing and building the CGC2 SOA. Equally important, they must accept the fundamental change from building vertical stovepipe information systems to horizontally integrated ones. SOA's modular horizontal integration unlocks the functionality and data trapped inside stovepipe systems and allows imaginative Coast Guard personnel to create new uses from existing components. The cloud at the top of Figure 24 graphically represents this "vision." The Architecture Loop continues with four sequential steps and their outputs. The first two steps incrementally produce the services; the final two steps provide continuous improvement. The Architecture Loop's steps and outputs are listed in Table 3.

Steps	**Outputs**
Design SOA	Set of Services
Build One Component	Functioning Component
Re-evaluate	Revised Business Processes and Revised Technical Processes
Adjust Vision, Strategy, and Concepts	Revised Vision, Strategy, and Concepts

Table 3. IDeA Architecture Loop – Steps and Outputs

Design SOA – The design process combines the stakeholders' visions, technical strategies, and desired end states. It produces many concepts represented in various forms: utility trees of quality attributes and scenarios, line diagrams of components and the relationships between them, and lists of required technical standards. Ultimately, service designers transform these concepts into distinct services with detailed descriptions of their functionality, interfaces, and interactions (with external systems and other services).

Build One Component – This step represents the Service Development Loop (SDL) that will be described in the following section. In this step, developers convert a description into a functioning service. This incremental SOA implementation generates test cases, metrics and measured qualities to verify that the service performs as described.

Integrate New Component Into SOA – This step integrates the newly created service into the SOA. Existing workflows and processes may need modification to incorporate the new component appropriately and maximize the benefits it provides.

Re-evaluate – This step in the Architecture Loop exists to capture lessons learned from building the last service. It begins the continuous improvement effort by assessing the new service's technical and business usefulness, or "operational performance." The technical review compares quality attribute levels (e.g., security, reliability, scalability, etc.) in the new service to those sought in the architecture. The SDL also evaluates each service. However, the SDL review focuses on the service's *internal* workings. In contrast, this technical review identifies architectural changes necessary to rectify problems and prevent similar shortcomings in future loops. The business review looks at the service's functionality and outputs to determine its fit within the workflow. This identifies modifications to the new service, and existing services, to improve overall performance.

Adjust Concepts – This step takes what you have learned and revises the concepts, vision, and technical strategy that shape the architecture. We expect each pass through the Architecture Loop will bring improved understanding of the architecture and implemented services. Stakeholders will have first-hand experience about what can be accomplished and how. Their improved understanding will likely lead to architectural changes, which restarts the loop at the "Design SOA" step.

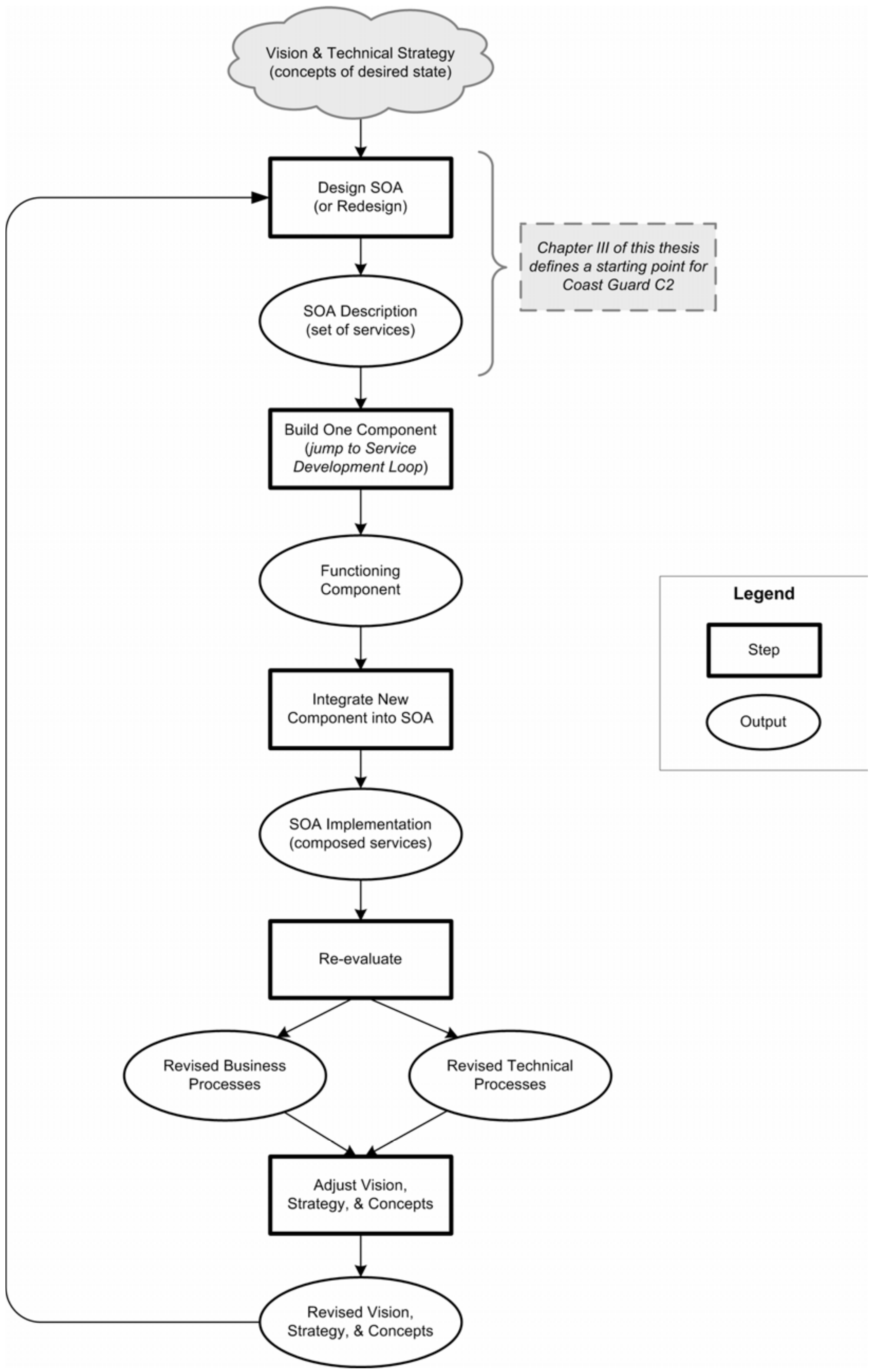

Vision & Technical Strategy (concepts of desired state)
Design SOA (or Redesign)
Chapter III of this thesis defines a starting point for Coast Guard C2
SOA Description (set of services)
Build One Component (jump to Service Development Loop)
Functioning Component
Legend
Step
Output
Integrate New Component into SOA
SOA Implementation (composed services)
Re-evaluate
Revised Business Processes
Revised Technical Processes
Adjust Vision, Strategy, & Concepts
Revised Vision, Strategy, & Concepts

Figure 24. IDeA Architecture Loop

3. Service Development Loop (SDL)

The SDL begins by describing *why* and *how* to invoke the service. The developers must clearly understand the contexts in which the service operates to implement it effectively. This goal and context awareness describes the “voice of the customer,” and the cloud at the top of Figure 25 graphically represents it. Table 4 lists the SDL steps.

Steps	**Outputs**
Identify Functionality and Quality Attributes	Unconstrained list of Functions and Quality Attributes
Develop Scenarios for Quality Attributes and Functionality	List of brief scenarios
Prioritize and Select	List of Quality Attributes and Functionalities to be implemented during current iteration
Identify External Interactions and Interfaces	Service Interface Descriptions
Identify Measures	List of metrics and success thresholds
Develop and Deploy Service (return to Architecture Loop)	Working service that performs required functionality with proper quality attributes
Measure and Evaluate	Service performance areas for future development and revision
Review and Adjust Process	Process improvements based on lessons learned

Table 4. IDeA Service Development Loop – Steps and Outputs

This loop contains eight steps, with a split after the “Develop and Deploy Service” step. To continue overall system development, you return to the Architecture Loop and continue that process with the newly created service, proceeding to develop the next component. The SDL moves on to measure, evaluate, improve, and evolve the current service as necessary. It also identifies ways to adjust the SDL process itself.

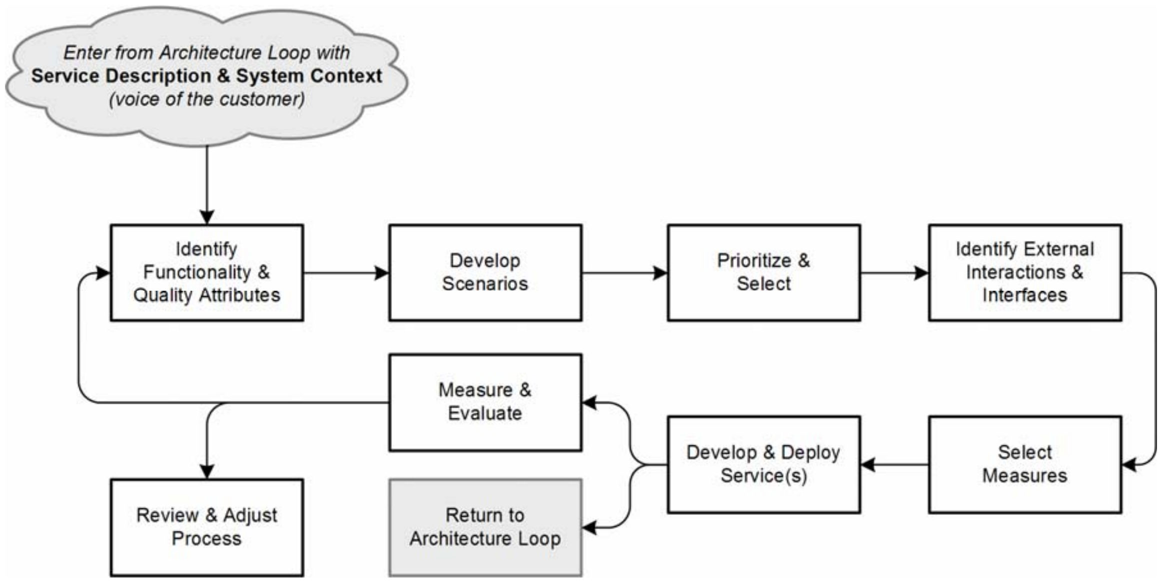

Enter from Architecture Loop with
Service Description & System Context
(voice of the customer)
Identify Functionality & Quality Attributes
Develop Scenarios
Prioritize & Select
Identify External Interactions & Interfaces
Measure & Evaluate
Develop & Deploy Service(s)
Select Measures
Review & Adjust Process
Return to Architecture Loop

Figure 25. IDeA Service Development Loop

Identify Functionality and Quality Attributes – The first step in the SDL transforms the initial service description, assumptions, and context about the system state into a detailed functionality list (actions and outputs) and relevant quality attributes. These items heavily influence the service's design. Stakeholders prioritize the functionality and quality attributes in the SDL's next steps. This process ensures the appropriate scope of work for the current loop iteration. The first iteration creates a fairly simple service, focusing on the most important quality attributes. Future iterations deliver increased complexity until they meet all service requirements.

Develop Quality Attributes and Functionality Scenarios – This step creates brief, precise scenarios that make the functionality and quality attributes concrete. These scenarios ensure the development team and stakeholders accurately understand what the new service does and how.

Prioritize and Select – The scenarios generated during the previous step and "voice of the customer" prioritize the functionality and quality attributes. The stakeholders and development team choose the scenarios to implement during the current SDL iteration. Because the scenarios directly correspond to functionality and quality attributes, everyone involved understands the service's requirements.

Identify External Interactions and Interfaces – The next step identifies the external services, systems, and data the service consumes to produce the desired output. Additionally, it specifies the incoming message type, content, and format, which define the service invocation methods. Similar details describe the service's output. These definitions specify the service's interfaces.

Select Measures – This step supports continuous improvement by identifying what aspects to measure to determine if the service provides the required functionality and quality attributes. Each measurement includes thresholds that clearly define success or failure.

Develop and Deploy Service – The previous five steps create a logical and understandable service definition. This step develops and deploys that service to provide the prioritized functionality and quality attributes, using the proper interfaces. Following deployment, we return to the Architecture Loop to continue that process. The SDL also continues with two more steps in the loop.

Measure and Evaluate – This step collects the measurements and evaluates them based on stated thresholds. This determines whether or not the service works as expected and meets the users' needs. Stakeholder feedback identifies new requirements for future development and revision. The SDL restarts at the "Identify" step to address existing defects or develop new requirements.

Review and Adjust Process – Continuous improvement also extends to the process used to develop the service. The development process trials and tribulations will result in "lessons learned," used to improve the SDL during future loops.

4. IDeA Conclusion

We believe our proposed two-loop method provides the Coast Guard with a flexible and incremental, design and implementation process. The IDeA method will immediately deliver useful services, and provide evolutionary architectural improvement. Each loop cycle will not only develop additional services but also allows us to improve our developmental methodology as we learn more about our needs and the technology.

C. ORGANIZE FOR SUCCESS

Designing and implementing a SOA should revolutionize the Coast Guard's information technology capabilities and infrastructure. The organizational impact can and should be equally as dramatic. Consider the transformation at Amazon.com in 2001. The online retail giant realized their existing monolithic application could not scale to meet future needs. Amazon implemented an SOA and organized their numerous development teams around the services within the SOA. Amazon's Chief Technical Officer (CTO) Werner Vogels describes the impact this approach had in the following quote:

> The services model has been a key enabler in creating teams that can innovate quickly with a strong customer focus. Each service has a team associated with it, and that team is completely responsible for the service—from scoping out the functionality, to architecting it, to building it, and operating it. … There is another lesson here: Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service. (Gray)

The Coast Guard operates in the *traditional* model described by Vogels. One group envisions each system, another designs it, and a third foists it on the user. Our traditional approach created our existing stovepipe applications that don't meet our current or future needs. We need to seize the opportunity that SOA provides and break this pattern by changing our system development organization to replicate Amazon's approach.

The first step in our organizational makeover designates the chief architect. This person must have a credible architectural vision and the ability to communicate it to others in a clear and convincing way. He or she will lead the architecture design team to produce the overall SOA[2], including guidance other development teams will follow. In addition, this team will act as the steering committee. The team will proactively manage the service development, deployment, and improvement efforts that occur during the IDeA method iterations. They will also identify the resources required to implement and maintain the CGC2 SOA. These resources include hardware, software, personnel, training, and other funding items.

Continuing to emulate Amazon's approach, we will have each existing organizational entity develop services within its own domain. These entities will form development teams that create, deploy, maintain and evolve services using the standards and guidance from the chief architect. For example, the Coast Guard Operations Systems Center (OSC) owns our databases and therefore should produce the data and enterprise business services. The Telecommunications and Information Systems Command (TISCOM) should produce network and security services and propose overall system policy standards. The Coast Guard Command and Control Engineer Center (C2CEN) should produce the operational tasking, geospatial display, and sensor monitoring services. These three commands would also collaborate to share lessons learned and propose modifications to the standards and polices that the chief architect establishes.

D. BEST PRACTICES AND WORST PRACTICES

This section continues our "learning from others" approach to implement the CGC2 SOA successfully. While researching and writing this thesis, we noticed several recurring suggestions that we should pay careful attention to, understand and use. The diagram below contains those fundamental "best

[2] We think the SOA presented in Chapter 3 provides an excellent starting point.

practices" and "worst practices" that various companies and individuals working in the SOA marketplace have identified.

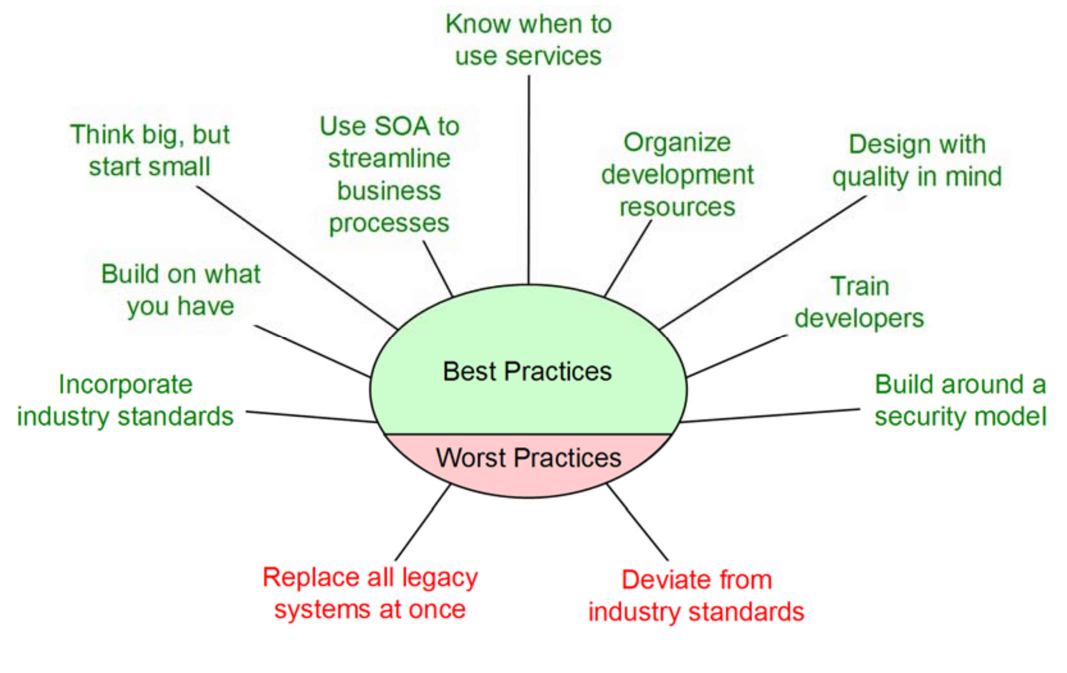

Know when to use services
Think big, but start small
Use SOA to streamline business processes
Organize development resources
Design with quality in mind
Build on what you have
Train developers
Incorporate industry standards
Best Practices
Build around a security model
Worst Practices
Replace all legacy systems at once
Deviate from industry standards

Figure 26. SOA Best Practices and Worst Practices

Know when to use services – This best practice requires that we explicitly define the extent to which we will use services. Using a Web service does not require an entirely new application architecture. SOA's loosely coupled design allows the limited addition of services without a negative impact on the remaining application architecture. (Erl 448) The corollary to this best practice advises us to "know when to avoid services." The "Technical Strategy" and "Design SOA" Architecture loop steps, introduced earlier in this chapter, apply these two best practices. We actively select what to create in each service development cycle. Services will not randomly spring up across the Coast Guard's enterprise architecture.

Think big but start small – This best practice appeared in almost everything we read. "Be selective. Don't start with a massive project that involves a cast of thousands. Think big but start with a small project. Focus on a project that can highlight the clear benefits of SOA like reworking a small set of key business processes to improve their flexibility." (Coticchia 8) The IDeA method fully supports this best practice. Starting small validates the architecture while giving the organization value, realized as usable services. We don't want to develop a collection of fragmented services. To avoid this we need to "create the architecture and deploy specific services in phases, perhaps focusing on one application domain at a time or choosing projects based on business urgency." (Gruman)

Replace all legacy systems at once – This worst practice states that migrating the entire enterprise to SOA in one large project "… is a recipe for disaster. Theoretically, it may seem like a good idea to jump right into SOA implementation, ripping out and replacing all existing systems at once. SOA technology is new, exciting and hugely beneficial, and it's easy to get carried away." (Foody 28,29) The Coast Guard's SOA implementation plan should migrate our entire enterprise to SOA over many years. The IDeA method's incremental, evolutionary approach avoids this worst practice.

Build on what you have – This best practice considers "… reusing legacy logic before replacing it. Web services can let you take advantage of what you already have through the use of adapters and service layers." (Erl 451) Each IDeA method iteration should reuse legacy application functionality and data wherever possible. For example, the AOPS database records asset employment data. It's a burden for the all the organizational levels to keep current. We can transform this database from a historical archive into the Coast Guard's assets status board.

Use SOA to streamline business processes – This best practice capitalizes on SOA's inherently flexible and interoperable model for hosting application functionality. SOAs provide an opportunity to rethink and improve business processes. The Coast Guard should grasp this opportunities to streamline its business processes. Continuing the AOPS example, the database update could be worked into every business process that tasks assets and impacts their employment category. This would streamline a currently disjointed work flow.

Incorporate standards – This best practice suggests using the industry Web service standards (W3C, OASIS) as the standards for the Coast Guard's SOA. "In an enterprise, this can potentially translate into a standardized system for navigating: application logic, integration architectures, corporate data stores, and parts of the enterprise infrastructure." (Erl 454) The initial Architecture Loop iteration should identify which standards will be used.

Deviate from industry standards – Modifying those industry standards to fit within current system configuration creates more problems than it solves. This worst practice occurs when people try to save time and money during the current development cycle. However, standards exist for a reason; modifying them can cause unintended, severe interoperability issues. Customizing standards requires special code at every affected service or node to function properly. This creates brittle connections and defeats the purpose of a loosely-coupled SOA. The Coast Guard should avoid this problem.

Build around a security model – "The functional design needs to be built upon the security model, not the other way around. Putting together a design, and perhaps even building a preliminary version of your [system] without serious consideration for the underlying security model is a common mistake." (Erl 463) This best practice recognizes that security often cannot be added to an architecture or application as an afterthought. The CGC2 SOA requires strong security. Therefore the Coast Guard must include it in the initial architecture design.

Design with quality in mind – "This has never been more important than in an SOA environment. Specifically for a development issue, quality must be designed into the product not inspected into it." (Coticchia 9) The Coast Guard must identify key quality attributes and then properly balance their trade-offs when designing the SOA. The utility tree in Chapter 3 provides a good starting point for the Coast Guard.

<u>Organize development resources</u> – This best practice groups development teams around logical business tasks. "A common mistaken during development projects is to have one team deliver the Web services and a different team develops the rest of the application. This approach may make sense, because you have each team working with technologies that they know how to use. It can make the resulting application seem disjointed and non-intuitive to the user." (Erl 465) Section C above embodies this best practice.

<u>Train developers</u> – This best practice ensures that designers and developers have the skills necessary to implement the Web services properly. (Erl 466) Software developers need to understand service-oriented principles and practices, as well as the Web services technical details. The Coast Guard should identify and provide the training each development team member requires. While this costs money, it pays big dividends.

E. SOA'S IMPACT ON QUALITY ATTRIBUTES

While creating the architecture description in Chapter III, we compiled a list of possible quality attributes. We ranked them based on our personal judgment about their importance to a Coast Guard command and control system architecture. Figure 27 below shows this ranking. We selected the top five quality attributes and used them to develop the utility tree in Chapter III (Figure 8). The other seven quality attributes certainly require some attention when developing a system based on this architecture. However, we feel that the top five would most influence the architectural design and should receive greater attention.

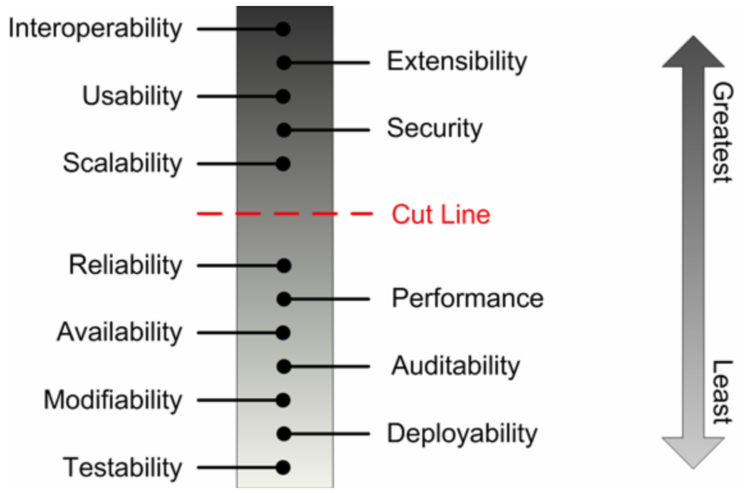


Figure 27. Quality Attribute Importance for CGC2 SOA

A September 2005 report from the Software Engineering Institute (SEI) addresses the positive and negative effects that an SOA has on a system's quality attributes. We assessed the material in that report and then ranked our top five quality attributes based on SOA's maturity level. Figure 28 shows the quality attributes well supported by SOA in green. The color red indicates quality attributes not well supported by current SOA technologies. We discuss the impact of these support concerns in the following paragraphs.

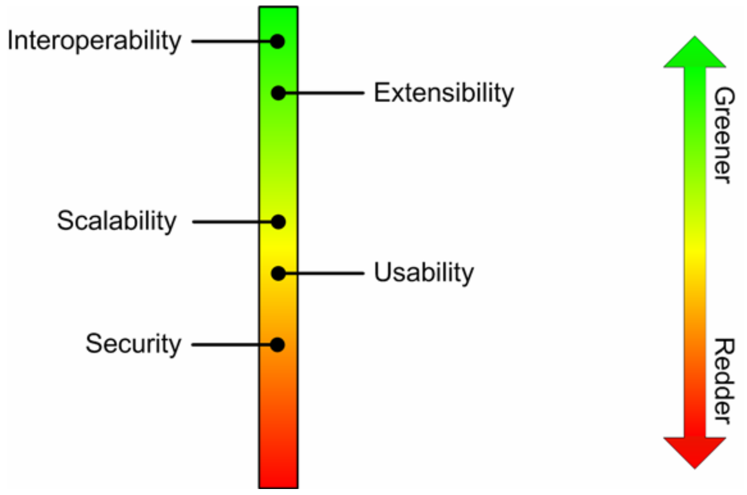


Figure 28. SOA Support for Quality Attributes

We have taken the quality attributes from Figure 28 above and quoted the appropriate sections from the SEI report in Table 5. The "status" column refers to SOA's maturity level for the quality attribute. "The color green indicates that there are known solutions for the SOA based on relatively mature standards and technology. The color yellow indicates that some solutions exist but need further research to prove their usefulness in handling the requirements for the quality attribute. The color red indicates that the standards and technology are immature and further significant effort is required to fully support the quality attribute within an SOA." (O'Brien, Bass, and Merson 22)

Quality Attribute	**Status**	**Summary**
Interoperability	Green	"Through the use of the underlying standards, an SOA provides good interoperability technology-wise overall, allowing services and applications built in different languages and deployed on different platforms to interact. However, semantic interoperability is not fully addressed. The standards to support semantic interoperability are immature and still being developed."
Security	Red	"The need for encryption, authentication, and trust within an SOA approach requires detailed attention within the architecture. Many standards are being developed to support security, but most are still immature. If these issues are not dealt with appropriately within the SOA, security could be negatively impacted."
Usability	Yellow	"Usability may decrease if the services within the application support human interactions with the system and there are performance problems with the services. It is up to the services users and providers to build support for usability into their systems."
Extensibility	Green	"Extending an SOA by adding new services or incorporating additional capabilities into existing services is supported within an SOA. However, the interface/formal contract must be designed carefully to make sure that it can be extended, if necessary, without causing a major impact on the service users."
Scalability	Yellow	"There are ways to deal with an increase in the number of service users and the increased need to support more requests for services. However, these solutions require detailed analysis by the services providers to make sure that other quality attributes are not negatively impacted."

Table 5. SOA Quality Attribute Impact (After: O'Brien, Bass, and Merson Table 1)

The Coast Guard's architects and implementers need to understand SOA's strengths and weakness when designing and building the CGC2 SOA. Our success depends on our ability to properly identify and address the limitations of the technology that support the architectural approach. The following paragraphs provide our specific responses for each quality attribute. However, we should closely monitor the emergence and improvement of industry standards and best practices for every quality attribute. Each successive IDeA loop cycle should selectively implement the standards and best practices appropriate to our needs.

Interoperability – SOA strongly supports this quality attribute. The SEI's concern about semantic interoperability can partially be addressed with appropriate information exchange data models.

Security – Several web service standards support confidentiality, authenticity, integrity, and non-repudiation. These standards have been updated with more mature versions since the SEI report appeared. Therefore we disagree with the "red" status and would classify it currently as yellow. This comment does not diminish the security problem's complexity. We will likely develop multiple approaches to meet the needs of users that have established various trust relationships. This quality attribute obviously must be approached architecturally, incrementally, and without excessive risk, delay, or simplifications that produce either an overly rigid system or an insecure one. Additionally, providing the Public Key Infrastructure (PKI) required to implement these standards will require careful consideration early in the IDeA process.

Extensibility – SOA also strongly supports this quality attribute. The IDeA method will enforce properly designed interfaces, enabling each service to be extended without negatively impacting users.

The decisions made by the architects and developers heavily influence the two remaining quality attributes. Implementing a certain industry standard will not provide required usability or scalability. However, the Amazon approach (develop services with the same people that provide the business functionality) will provide the proper developer motivation and perspective to deliver these quality attributes.

Usability – Developers address undocumented or vaguely defined performance dimensions when they understand the unique user requirements for each service. This inherent awareness of user needs goes a long way to addressing usability.

Scalability – Amazon, AT&T and British Telecom all have extremely large-scale SOAs. It's very important to properly identify and address scalability requirements early in the IDeA process. However the Coast Guard's scalability concerns won't exceed those of industry-leading SOA adopters.

F. CONCLUSION

Our answer to the second thesis question proposed an iterative method to design and implement the architecture, logically organize the development teams, and learn from industry best practices. Taken as a whole, this collection provides the Coast Guard with a solid foundation to begin designing and implementing the CGC2 SOA.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

The Coast Guard Commandant recently mandated that we shift our IT infrastructure to an SOA. To achieve this, we must carefully determine *what* our SOA will look like and *how* we will successfully build it. This thesis provides the foundation for both. In Chapter two, we began by defining SOA concepts and technologies. Many SOA technical standards and enabling software remain immature, but the industry improves them at a quick and steady pace. We must monitor their continued advances and adjust our SOA accordingly. In Chapter three we described how the Coast Guard could implement a command and control SOA. We included the services, their interactions, the data, and the quality attributes the architecture must address. Chapters two and three thus crystallize *what* the Commandant has mandated.

SOA forces us to change the way we conceive and implement our information technology, shifting from vertical stove-piped systems to horizontally integrated ones. In Chapter four, we introduced a two-loop method for incrementally building the SOA in a way that evolves from the present towards the constantly moving, desired future state. Focusing on short, clearly defined implementation cycles allows us to incorporate lessons learned to improve the architecture, its components, and the way we create them. We reviewed several industry best practices and common pitfalls, including a recommended organizational alignment deemed crucial to Amazon's successful SOA. Finally, we discussed the impact an SOA has on our chosen quality attributes. Chapter four thus answers *how* we should meet the Commandant's mandate.

We began our thesis research by reading several white papers, from companies selling SOA software products or consulting services to implement SOAs. These papers described SOA solving every computer system integration and data sharing problem in existence. Our further research and practical experience with the Comprehensive Maritime Awareness JCTD proved otherwise. Contrary to the advertisements, we cannot simply purchase an SOA from a vendor or order the Coast Guard's IT staff to create one. The Coast Guard's requirements to reach mobile platforms further complicate matters. We can not rely on ample internet bandwidth to extend the SOA to boats, aircraft, and cutters. After finishing our research and thesis work, we conclude that SOA does not provide "the answer to everything." Nevertheless, we believe that SOA, properly managed, can deliver tremendous benefit to the Coast Guard. We think that Coast Guard can and should use SOA to revolutionize our command and control.

B. RECOMMENDED FUTURE RESEARCH

While researching and writing this thesis we identified several items that future NPS thesis students can develop further. We list them below.

1. Coast Guard Data Models

In this thesis we created basic data models for demonstration purposes only. A graduate student could devote his or her thesis to researching and developing the data model for the entire CGC2 SOA or merely develop the most valuable data models for near-term implementation. Either way, this difficult but crucial effort would require close work with several Coast Guard entities.

2. Planning Services Based on MHS-OPS

The MHS-OPS developers implemented a useful HLS planning and tasking tool for homeland security. Unfortunately, they built another stovepipe system. We recommend a thesis student service-enable the MHS-OPS functionality, at the proper level of abstraction, so all Coast Guard mission areas can use it.

3. Operations Watchstander Console

The typical Coast Guard command center watchstander has to manage multiple computer screens connected to many different computer systems. We envision a single integrated composite application to manage all watchstander computing and information management tasks. It should manage operational tasking, checklists and watch logs and ensure the watchstander complies with

Coast Guard regulations and local SOPs. A graduate student could research and develop all or part of the composite application.

4. PKI for SOA

The Coast Guard must address service and security early in its SOA development. We need appropriate PKI to issue credentials to users and services (including end systems) operating in the SOA. The solution to this non-trivial problem will address an important quality attribute and will be reused in all Coast Guard SOAs. A graduate student could research the security and PKI aspects of current DoD SOA implementations and propose a Coast Guard specific solution. Regardless of graduate research, the Coast Guard needs to make this an action item for development funding and implementation.

5. XMPP for Coast Guard Command and Control

The United States Marine Corps recently adopted XMPP as its standard instant messaging protocol. XMPP can provide much more than chat and instant messaging within an SOA. NPS faculty and students have researched using XMPP for passing data (e.g., tracks) between battlespace nodes. We recommend researching XMPP as a means to pass operational tasks (e.g., search patterns) between Coast Guard units. The data needs to have proper formatting for easy transfer into cutter, boat, and aircraft navigation systems.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

"ABCs of SOA." *CIO.com.* 15 Jun. 2006. 13 Mar. 2007. <http://www.cio.com/abcs/soa_abc.html?action=print>

Allen, Thad W. "Commandant's SITREP 2 - First 100 Days" 20 Dec. 2006. <http://www.uscg.mil/comdt/all%5Fhands/message5.asp>

America's Maritime Guardian. Coast Guard Publication 1. Revision 1 Jan. 2002.

"Average Day." *Coast Guard Fact File.* United States Coast Guard. 20 Dec. 2006. <http://www.uscg.mil/hq/g-cp/comrel/factfile/index.htm>

Bayne, Jay S. *Creating Rational Organizations: Theory of Enterprise Command and Control*. Cafepress.com, 2006.

Clements, Paul, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies.* Boston, MA: Addison-Wesley, 2002.

Command Center Program Manual. Coast Guard Commandant Instruction Manual (draft). 13 Nov. 2006.

Common Operational Picture, Operational Requirements Document. USCG Common Operational Picture Working Group. Ver 1.0. 04 Aug. 2005.

Coticchia, Greg. "The Seven Secrets of SOA Success." *SOA Web Services Journal* 6.7 (2006): 8-9. 01 Nov. 2006. <http://webservices.sys-con.com/read/250498.htm>

DoDAF Deskbook. DoD Architecture Framework Working Group. Ver 1.0. 09 Feb. 2004.

"Department Subcomponents and Agencies." *Department of Homeland Security Web Site.* U.S. Department of Homeland Security. 08 Mar. 2007. <http://www.dhs.gov/xabout/structure/index.shtm>

Erl, Thomas. *Service-oriented Architecture: A Field Guide to Integrating XML and Web Services.* Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2004.

Gabriel, Jim. "Best Practices in Integrating Data Models for SOA." *SOA World Magazine*. 02 Feb. 2005. 27 Feb. 2007. <http://webservices.sys-con.com/read/48031.htm>

Gray, Jim. “A Conversation with Werner Vogels.” *ACM Queue*. Vol. 4, No. 4. May 2006. 13 Mar. 2007. <http://www.acmqueue.com/modules.php?name=Content&pa=printer_friendly&pid=388&page=1>

Gruman, Galen. “SOA’s True Challenge – It Ain’t Technology” *CIO.com*. 01 May 2006. 10 Nov. 2006. <http://www.cio.com/archive/050106/et_main.html?action=print>

Hamilton, Alexander. “The Federalist (No. 12), The Utility of the Union In Respect to Revenue.” *The New York Packet.* 27 Nov. 1787. 20 Dec. 2006. <http://www.constitution.org/fed/federa12.htm>

Hayes-Roth, Frederick. *Hyper-Beings: How Intelligent Organizations Attain Supremacy through Information Superiority.* Booklocker.com, 2006.

Hutchins, Jeffery. “Enabling Oracle Integration B2B and Oracle BPEL Process Manager Interoperability.” *Oracle SOA Suite Best Practices.* Dec. 2006. Oracle Technology Network. 23 Feb. 2007. <http://www.oracle.com/technology/tech/soa/soa-suite-best-practices/b2b-bpel-integration.html>

Kurzweil, Ray. *The Age of Spiritual Machines*. New York: Viking, 1999.

Lau, Yun-Tung. “Service-Oriented Architecture and the C4ISR Framework.” *CrossTalk.* Sep. 2004. 11-14. 15 Jan. 2007. <http://www.stsc.hill.af.mil/crosstalk/2004/09/0409lau.html>

“Missions.” *Coast Guard Web Site.* United States Coast Guard. 20 Dec. 2006. <http://www.uscg.mil/top/missions/>

O’Brien, Liam, Len Bass, and Paulo Merson. “Quality Attributes and Service-Oriented Architectures.” *Software Engineering Institute Technical Note.* CMU/SEI-2005-TN-014. Sep. 2005. <http://www.sei.cmu.edu/publications/documents/05.reports/05tn014.html>

“Published Software Architecture Definitions.” *Software Engineering Institute.* 28 Jan. 2007. <http://www.sei.cmu.edu/architecture/published_definitions.html>

“SAML.” *oasis-open.org.* 15 Feb. 2007. 08 Mar. 2007. <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security>

“Software Architecture Glossary.” *Software Engineering Institute.* 21 Dec. 2006. < http://www.sei.cmu.edu/architecture/glossary.html>

"Stovepipe System." *Component Software Glossary*. 03 Jan. 1997. Object Services and Consulting, Inc. 21 Dec. 2006. <http://www.objs.com/survey/ComponentwareGlossary.htm#StovepipeSystem>

"The SOA Vision." *SOA Vision.com*. 03 Jan. 2007. <http://www.soasystems.com/soa1.asp>

"Web Services Wrapper." *Actional.com.* 08 Mar. 2007. <http://www.actional.com/resources/whitepapers/SOA-Worst-Practices-Vol-I/Web-Services-Wrapper.html>

"WS-Addressing." *w3.org.* 03 Aug. 2004. 08 Mar. 2007. <http://www.w3.org/Submission/2004/05/>

"WS-Coordination." *oasis-open.org.* 08 Feb. 2007. 08 Mar. 2007. < http://docs.oasis-open.org/ws-tx/wscoor/2006/06/wstx-wscoor-1.1-rddl-200702.htm>

"WS-Eventing." *Ibm.com.* Aug. 2004. 08 Mar. 2007. < ftp://www6.software.ibm.com/software/developer/library/ws-eventing/WS-Eventing.pdf>

"WS-Federation." *Networkworld.com.* 08 Mar. 2007. <http://www.networkworld.com/details/6284.html>

"WS-Manageability." *Ibm.com.* 10 Sep. 2003. 08 Mar. 2007. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-manage/ws-manage.pdf>

"WS-Notification." *oasis-open.org.* 01 Oct. 2006. 08 Mar. 2007. <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn>

"WS-ReliableMessaging." *oasis-open.org.* Aug. 2006. 08 Mar. 2007. <http://docs.oasis-open.org/ws-rx/wsrm/200608/wsrm-1.1-rddl-200608.html>

"WS-SecureConversation." *oasis-open.org.* 29 Nov. 2006. 08 Mar. 2007. <http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-spec-cs-01.htm>

"WS-Security." *oasis-open.org.* 01 Feb. 2006. 08 Mar. 2007. <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>

"WS-SecurityPolicy." *oasis-open.org*. 08 Dec 2005. 08 Mar. 2007. <http://www.oasis-open.org/committees/download.php/15979/oasis-wssx-ws-securitypolicy-1.0.pdf>

"WS-Trust." *oasis-open.org.* 06 Sep. 2006. 08 Mar. 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-spec-cd-01.html

www.niem.gov. 01 Mar. 2007. National Information Exchange Model. 02 Mar. 2007. <http://www.niem.gov/index.php>.

"XML." *Bytowninternet.com.* 08 Mar. 2007. <http://www.bytowninternet.com/glossary#xml>

BIBLIOGRAPHY

Allen, Thad W. State of the Coast Guard Address. The Renaissance Hotel, Washington, DC. 13 Feb. 2007 01 Mar. 2007. <http://www.uscg.mil/comdt/speeches/docs/transcript.pdf>

BEA White Paper. Jul. 2005. *Domain Model for SOA.* BEA Systems, Inc. 06 Dec. 2006. <http://bea.com/framework.jsp?CNT=research_whitepapers.htm&FP=/content/solutions/soa/library/>

BEA White Paper. Jun. 2005. *The Advent of a New Service Infrastructure.* BEA Systems, Inc. 06 Dec. 2006. <http://bea.com/framework.jsp?CNT=research_whitepapers.htm&FP=/content/solutions/soa/library/>

BEA White Paper. *The Integration Journey – A Field Guide to Enterprise Integration for SOA.* BEA Systems, Inc. 06 Dec. 2006. <http://bea.com/framework.jsp?CNT=research_whitepapers.htm&FP=/content/solutions/soa/library/>

BEA White Paper. *The Move Toward Shared Services.* BEA Systems, Inc. 06 Dec. 2006. <http://bea.com/framework.jsp?CNT=research_whitepapers.htm&FP=/content/solutions/soa/library/>

BEA White Paper. Aug. 2005. *Turning IT Vision Into Business Value.* BEA Systems, Inc. 06 Dec. 2006. <http://bea.com/framework.jsp?CNT=research_whitepapers.htm&FP=/content/solutions/soa/library/>

Besemer, David. "Keep to the Original Intent of SOA." *SOA Web Services Journal* 6.11 (2006): 36-37. 05 Jan. 2007 <http://webservices.sys-con.com/read/314107.htm>

Bjorklund, Gus. "Did You Know There's a 'C' in SOA?" *SOA Web Services Journal* 6.9 (2006): 36-38. 01 Nov. 2006. <http://soa.sys-con.com/read/284584.htm>

Booth, David et al. "Web Services Architecture." *World Wide Web Consortium*. 11 Feb. 2004. 08 Mar. 2007. <http://www.w3.org/TR/ws-arch/>

Chappell, David A. *Enterprise Service Bus.* Sebastopol, CA: O'Reilly, 2004.

Croom, Charles E., Jr. “Service-Oriented Architectures in Net-Centric Operations.” *CrossTalk.* Jul. 2006. 13-15. 15 Jan. 2007 <http://www.stsc.hill.af.mil/crosstalk/2006/07/0607croom.html>

Daconta, Michael C., Leo J. Obrst, and Kevin T. Smith. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management.* Indianapolis, IN: Wiley Publishing, 2003.

Erl, Thomas. *Service-Oriented Architecture: Concepts, Technology, and Design.* Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2005.

Foody, Dan. “SOA Worst Practices.“ *SOA Web Services Journal* 6.11 (2006): 28-30. 05 Jan. 2007. <http://webservices.sys-con.com/read/314095.htm>

Foody, Dan. “The Challenges of SOA.“ *SOA Web Services Journal* 6.9 (2006): 8-9, 24. 05 Jan. 2007. <http://webservices.sys-con.com/read/284550.htm>

Hayes-Roth, Frederick, and Daniel Amor. *Radical Simplicity: Transforming Computers Into Me-Centric Appliances.* Upper Saddle River, NJ: Prentice Hall PTR, 2002.

InfoWorld White Paper. *From Pilot to Payoff: Service-Oriented Architecture Hits Its Stride.* BEA Systems, Inc. 21 Nov. 2006. <http://bea.com/framework.jsp?CNT=research_whitepapers.htm&FP=/content/solutions/soa/library/>

“Intro to Software Quality Attributes.” *SoftwareArchitectures.com.* 12 Jan. 2007. <http://www.softwarearchitectures.com/one/Designing+Architecture/78.aspx>

Jackson, Joab. “At Your Service.” *Government Computer News.* 24 Apr. 2006. 10 Nov. 2006. <http://www.gcn.com/print/25_9/40462-1.html>

Kodali, Raghu R., “What is Service-Oriented Architecture?” *JavaWorld*. 13 Jun. 2005. 08 Jan. 2007. <http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html>

Linthicum, David S. “SOA and Data Integration.“ *SOA Web Services Journal* 6.8 (2006): 14, 17. 10 Nov. 2006. <http://webservices.sys-con.com/read/275055.htm>

Maritime Law Enforcement Manual. Coast Guard Commandant Instruction Manual 16247.1C. Revision Aug. 2003.

Martens, China. “Users Offer SOA Advice: Start Small.” *IDG News Service.* 10 Nov. 2006. <http://www.cio.com/blog_view.html?CID=26583>

McComb, Dave. *Semantics in Business Systems: The Savvy Manager's Guide: the Discipline Underlying Web Services, Business Rules, and the Semantic Web.* San Francisco, CA: Morgan Kaufmann Publishers, 2004.

McGovern, David. *Embracing SOA: The Benefits of Integration Independence.* TIBCO Software Whitepaper. 08 Jan. 2007. <http://www.tibco.com/solutions/soa/resource_library.jsp>

Narang, Sanjay. "Web Services, WS-* Specifications, and Interoperability." *SOA Web Services Journal* 6.11 (2006): 20-26. 05 Jan. 2007. <http://opensource.sys-con.com/read/314083.htm>

Pasley, James. "The ESB in Your SOA." *SOA Web Services Journal* 6.11 (2006): 16-18. 05 Jan. 2007. <http://webservices.sys-con.com/read/314082.htm>

Rhody, Sean. "The SOA Dichotomy." *SOA Web Services Journal* 6.8 (2006): 5. 02 Nov. 2006. <http://webservices.sys-con.com/read/275011.htm>

Sagar, Ajit. "How Much SOA?" *SOA Web Services Journal* 6.8 (2006): 6. 02 Nov. 2006. <http://webservices.sys-con.com/read/275038.htm>

Shaffer, Dave. "Best Practices for Building SOA Applications (part 1)." *SOA Web Services Journal* 6.8 (2006): 34-37. 02 Nov. 2006. <http://webservices.sys-con.com/read/275111.htm>

Shaffer, Dave "Best Practices for Building SOA Applications (part 2)." *SOA Web Services Journal* 6.9 (2006): 48-50. 02 Nov. 2006. <http://webservices.sys-con.com/read/284591.htm>

Sun White Paper. Mar. 2006. *The SOA Platform Guide: Evaluate, Extend, Embrace..* Sun Microsystems. 12 Jan. 2007. <http://www.sun.com/software/whitepapers/index.xml#2>

Sun White Paper. Mar. 2006. *Systematic Development and the Serve Oriented Architecture.* Sun Microsystems. 12 Jan. 2007. <http://www.sun.com/software/whitepapers/index.xml#2>

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. U.S. COAST GUARD ORGANIZATIONAL RELATIONSHIPS

A. WITHIN THE FEDERAL GOVERNMENT

As a component within the Department of Homeland Security, the Coast Guard "protects the public, the environment, and U.S. economic interests—in the nation's ports and waterways, along the coast, on international waters, or in any maritime region as required to support national security." ("Department Subcomponents and Agencies")

B. WITHIN THE COAST GUARD

The Coast Guard has divided its operational commands into geographic zones, with the Atlantic Area and Pacific Area commanders reporting to the Commandant. Figure 29 shows the Coast Guard's top level operational command structure.

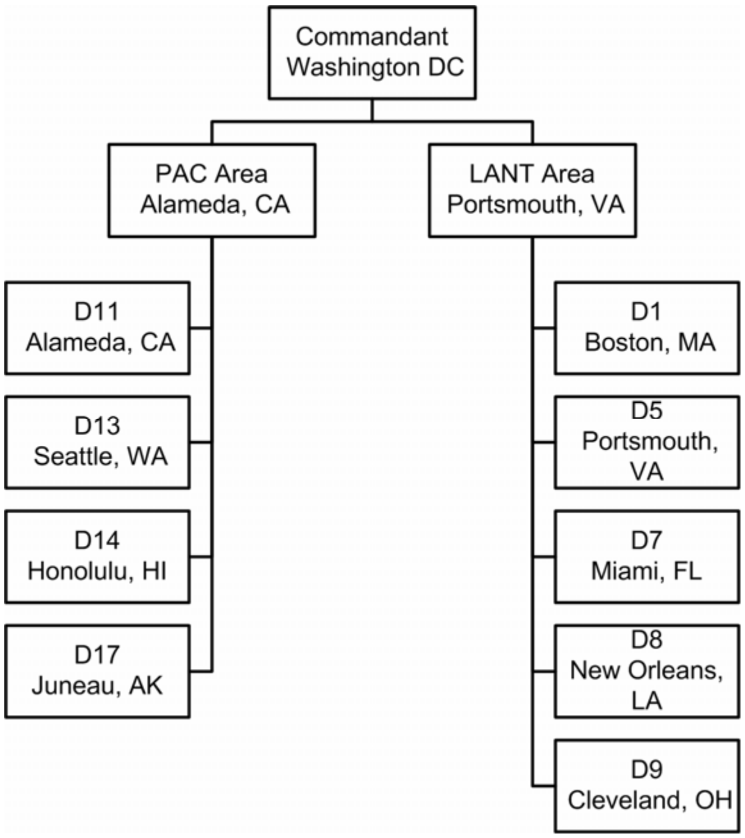


Figure 29. Coast Guard Operational Chain of Command

Both Areas are divided into Districts that cover several hundred miles of coastline. They generally correspond to a geographic region, for example the First Coast Guard District encompasses New England, stretching from Maine through New Jersey.

Each District is sub-divided into Sectors, shown in the Figure 30. Each Sector has several operational units under its command.

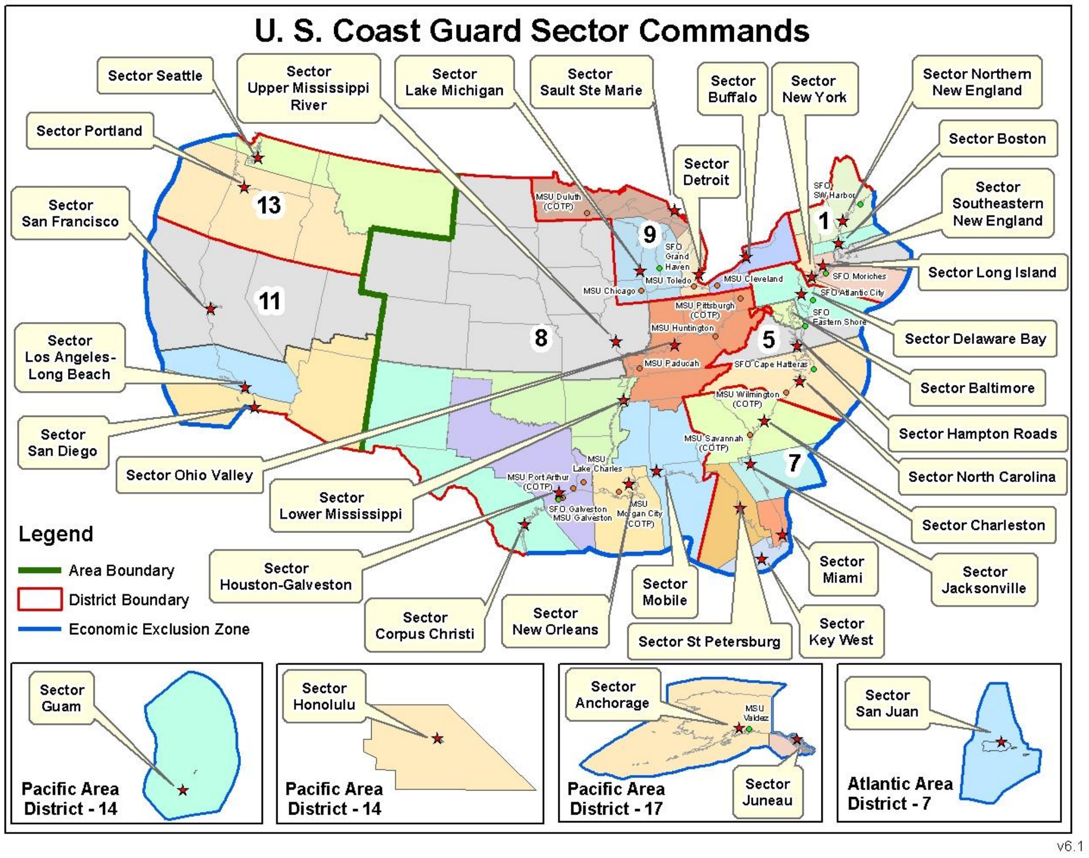


Figure 30. U.S. Coast Guard Sector Commands (From: *Command Center Program Manual* Figure 2-1-1)

C. WITHIN THE SECTOR

Each Sector command center (CC) performs the duties shown in Figure 31 and described below:

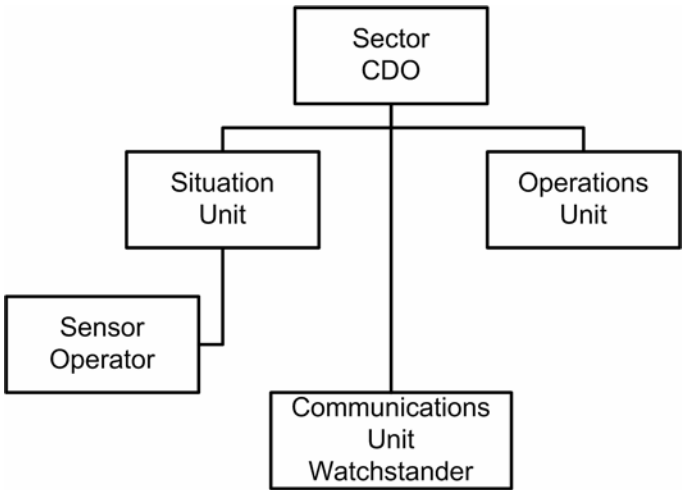


Figure 31. Sector Command Center Duties (After: *Command Center Program Manual* Figure 2-2-2)

CDO: "The CDO is responsible for the performance of the watch in the execution of its primary functions and ensuring proper coordination of operational plans for a specific operational period."

Situation Unit: "The Situation Unit is primarily responsible for monitoring the AOR, tracking the activities and readiness of blue forces, collecting and fusing of important information, and developing the local tactical picture."

Operations Unit: "The Operations Unit is responsible for the planning and execution of incident response missions conducted within the AOR. At the different levels of CCs, these responsibilities may translate into different positions. For example, some District and Sector CCs may have a staffed Law Enforcement watch position because of the elevated operational tempo (OPTEMPO) in LE cases within the AOR. Others rely on an on-call Law Enforcement Duty Officers (LEDOs) for SME guidance during LE cases. Additionally, District and Area CCs may elect to assign an officer with LEDET, MSST, or MSRT experience to plan and monitor use of Special Missions assets."

Comms Unit: "The Comms Unit is responsible for monitoring required voice frequencies, maintaining communication guard requirements, and, as directed, executing tactical communication for response operations.” (*Command Center Program Manual* 55-56)

D. CONCLUSION

The Coast Guard Commandant recently announced changes to our command structure. The information in this appendix, particularly at the District and Area level will change in the near future. However, we feel that this appendix allows non-Coast Guard readers to understand our current organizational layout as it is discussed throughout the thesis.

APPENDIX B. COMMUNICATIONS INTEROPERABILITY

A. NETWORK-CENTRIC

The Coast Guard cannot implement Network Centric Operations (NCO) without a network, or more specifically a network with the right capabilities that connects the right people and things. It is extremely unlikely that a single program or platform can bring about a transformation of the Coast Guard's IT infrastructure to close the gap between what we have and what we need. Therefore, it is imperative that we, as an organization, share a common view of these requirements and ensure that all current and future acquisitions work toward a common goal. In short, we need a consistent procurement approach that delivers components that meet these requirements.

Current command and control methodology places the various sense, decide, and act systems at the center of the diagram. Network Centric Operations places *the network* at the center as the key enabling technology. This perspective views all attached devices, services, and systems as nodes on the network. To enable NCO, the network definition includes: internet protocol (IP) routing, support for public key infrastructure (PKI), and message prioritization options for quality of service (QoS) beyond "best effort" delivery. It also means that all nodes and devices have three types of interfaces: network interfaces, management interfaces, and messaging interfaces.

While the Coast Guard has generally done well acquiring network infrastructure with the internet protocol (IP) data network reaching most users. However, the fatal exception occurs at the "last mile." The IP network has not been extended out to small boats, patrol boats, and aircraft. Further, many of the operational end systems such as radars on large cutters have not been attached to the unit's LAN. The Coast Guard's lag to extend network reach to mobile assets stems from the complexity of the problem itself. The combination of the operating environment, distance, mobility, RF spectrum, and availability of inexpensive hardware/software products all conspire to make this difficult. There are some initial solutions out in the marketplace but many have not reached the level of maturity to allow the Coast Guard to implement them on a large scale. The Naval Postgraduate School (NPS) has done research and experiments using products conforming to IEEE 802.16 standard, but the experimentation has not shown that it meets all Coast Guard requirements. However, 802.16 is a rapidly emerging technology that may provide a viable solution to the “last mile” problem. Regardless of the how it gets done, the Coast Guard needs to send and receive data from its boats, cutters, and aircraft.

B. REQUIRED NETWORK CAPABILITIES

Before diving into network requirements it is prudent to define what a network is. The network is all of the plumbing that connects the end systems together. The plumbing consists of the switches, routers and all of the connecting wiring.

These requirements apply to the network on two levels, the local area network (LAN) within one unit (boat, cutter, aircraft, boarding team, etc.) and then the wide area network (WAN) connecting all units. The LAN connects all “sense, decide, act” systems within a unit, examples include: GPS, radar, radios, and cameras. A router connects the single unit to all other units via the WAN. In addition to the necessary bandwidth to transmit the messages required by the SOA, both the LAN and WAN must have the following:

1. Availability

Availability is critical to effective network communications. When we say availability we mean that you must not have a single point of failure. Redundant data paths may not be feasible in every situation but it is important along crucial communication routes. Capacity is another important and often overlooked aspect of availability. A working link that cannot handle the bandwidth requirements is the same as no link at all for most users.

2. Quality of Service (QoS)

Quality of Service control mechanisms provide different priority to different users or data flows. QoS guarantees become more important when the network capacity is limited, like the Coast Guard's "last mile" reach to mobile assets. This vital concept has not been addressed with the CGDN, which relies on traditional "best effort" that performance is dependent on the current network traffic load. However, as services are added and users rely on data feeds for mission critical situations, prioritization of packets will become necessary. For example messages that contain the details of a search pattern must be received quickly and in tact. Effectively balancing network load will be critical for low-bandwidth users (small boats and aircraft) who will rely on QoS.

3. Public Key Infrastructure (PKI)

Traditionally the Coast Guard has relied on transport security, or encrypting the data pipes, for information security.

However, in a SOA the messages themselves require more robust security capabilities, especially when interacting with data sources outside the Coast Guard network. The Coast Guard must also interact with agencies outside the Federal Government and the Military. This adds a level of complexity to the security equation.

Public Key Infrastructure provides the foundation for multiple security qualities including:

- Authenticity – the sender's identification is correct
- Confidentiality – authorized users are granted access to information and unauthorized users are denied access.
- Integrity – the information has not been tampered with during the transit between sender and receiver.
- Non-repudiation – the sender can not refute sending the message and the receiver can not refute delivery.

With PKI in place, the services that make up the SOA can request and provide the required security qualities. PKI provides encryption at the source so that unprotected data never touches the network and it stays protected until it reaches the destination.

4. SNMP for Remote Management

If the network is the enabling technology at the center of our operations, then we need to properly monitor and manage its performance. Network management systems (NMS) use the Simple Network Management Protocol (SNMP) to monitor network-attached devices for conditions that warrant administrative attention. SNMP uses software agents that reside on the network devices to translate local management information into common format.

SNMP agents need to be provisioned onto all the components of the information systems by default. The NMS monitors the network and generates alarms when conditions are not within defined parameters. This is extremely important for mobile users with fragile connections.

C. WEB SERVICES STACK

The final requirement is a method for distributing the data. This thesis describes a services oriented architecture (SOA) for command and control where the data is exchanged in XML formatted messages between services. The World Wide Web Consortium’s Web Services Architecture Working Group defined technical standards to ensure interoperability for SOAs. The Working Group divided these standards into the following six areas: processes, descriptions, messages, communications, security and management: Figure 32 shows a modified version of their Web Services Architecture Stack diagram.

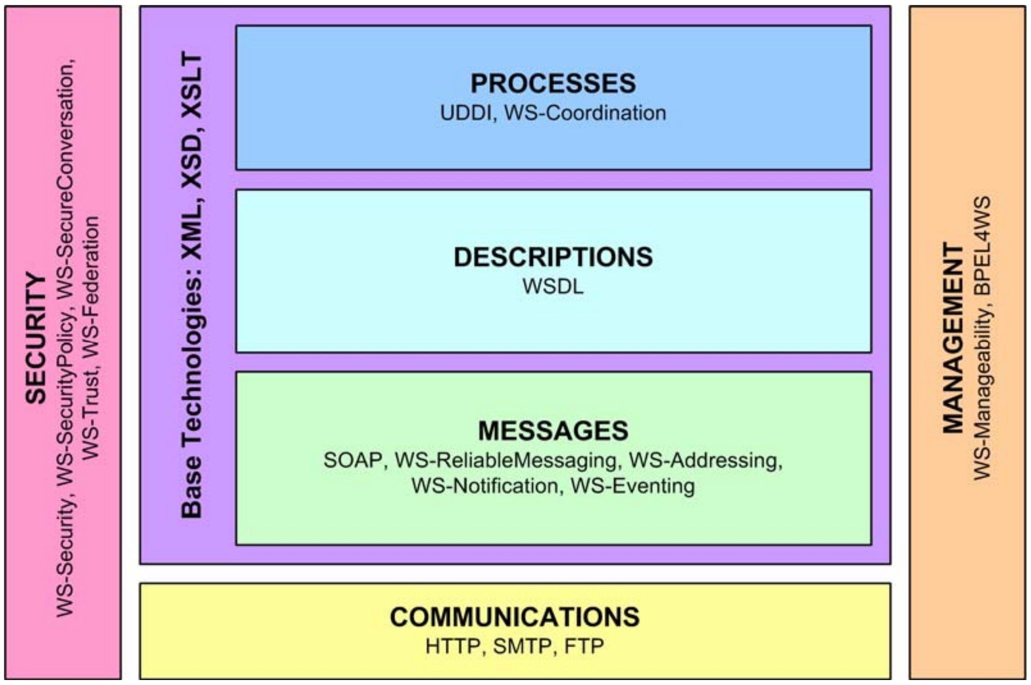

SECURITY
WS-Security, WS-SecurityPolicy, WS-SecureConversation, WS-Trust, WS-Federation
Base Technologies: XML, XSD, XSLT
PROCESSES
UDDI, WS-Coordination
DESCRIPTIONS
WSDL
MESSAGES
SOAP, WS-ReliableMessaging, WS-Addressing, WS-Notification, WS-Eventing
COMMUNICATIONS
HTTP, SMTP, FTP
MANAGEMENT
WS-Manageability, BPEL4WS

Figure 32. Web Services Architecture Stack (After “Web Services Architecture” Figure 3-1)

1. Process Layer

The Process layer describes how providers publish services and requestors/consumers discover them.

2. Description Layer

The Description layer describes how the service provider communicates the specifications for invoking the Web service to the service requestor

3. Messages Layer

The Messages layer describes how the services pass information in the form of a message

4. Communications Layer

The Communications layer describes how messages are physically transported across the network.

5. Security

Security occurs at all layers in the stack and it provides authenticity, integrity, confidentiality, and non-repudiation.

6. Management

Management, like Security, occurs across all layers in the stack. Management provides methods for monitoring and managing services and business processes.

APPENDIX C. SEARCH PATTERN WEB SERVICE SOURCE CODE

A. CODE OVERVIEW

These Java classes provide the search pattern Web service. The SearchPattern class is the actual Web service code. It calls methods from the Search class. The methods in the Search class are ParallelSearch, SectorSearch and SquareSearch. The Search class calls methods from the Nav class to calculate distance and bearing. The Position class is the instantiation class for the position object that is the core component of all searches. The WSDL provides the necessary information to the client so that it may consume the service. It defines which functions may be called and the parameters that are required to call them.

B. SEARCH PATTERN CLASS

```
/*
 * SearchPattern.java
 *
 * Created on November 20, 2006, 1:08 PM
 *
 */

package mil.uscg.nav;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 *
 * @author Bob Creigh
 */
@WebService()
public class SearchPattern {
    /**
     * Web service operation
     */
    @WebMethod
    public Object parallelSearchWS(@WebParam(name = "lat") double lat, @WebParam(name =
"lon") double lon, @WebParam(name = "length") double length, @WebParam(name = "width")
double width, @WebParam(name = "ma") double ma, @WebParam(name = "ts") double ts) {
        Search search = new Search();
        String  posit = lat + "\t" + lon + "\n";
        int i = 0;
        search.ParallelSearch(lat, lon, ma, width, length, ts);
        for(i=0;i<search.size();i++){
           posit = posit + (search.get(i).getLat()) + "\t" + (search.get(i).getLon()+
"\n");
        }

        return posit;
    }

    /**
     * Web service operation
     */
    @WebMethod
    public Object sectorSearchWS(@WebParam(name = "lat") double lat, @WebParam(name =
"lon") double lon, @WebParam(name = "theta") double theta, @WebParam(name = "radius")
double radius, @WebParam(name = "crs") double crs) {
        Search search = new Search();
        String  posit = lat + "\t" + lon + "\n";
        int i = 0;
        search.SectorSearch(lat, lon, theta, radius, crs);
        for(i=0;i<search.size();i++){
           posit = posit + (search.get(i).getLat()) + "\t" + (search.get(i).getLon()+
"\n");
        }

        return posit;
    }

    /**
     * Web service operation
     */
    @WebMethod
    public Object squareSearchWS(@WebParam(name = "lat") double lat, @WebParam(name =
"lon") double lon, @WebParam(name = "sqCount") double sqCount, @WebParam(name = "ts")
double ts, @WebParam(name = "crs") double crs) {
        Search search = new Search();
        String  posit = lat + "\t" + lon + "\n";
        int i = 0;
        search.ExpSquareSearch(lat, lon, sqCount, crs, ts);
        for(i=0;i<search.size();i++){
           posit = posit + (search.get(i).getLat()) + "\t" + (search.get(i).getLon()+
"\n");
        }

        return posit;
    }

}
```

C. SEARCH CLASS

```
/*
 * Search.java
 *
 * Created on September 8, 2006, 10:23 PM
 *
 */

package mil.uscg.nav;
import java.util.*;

/**
 *
 * @author Bob Creigh
 */
public class Search {
List<Position> searchList = new ArrayList<Position>();

    /** Creates a new instance of Search */
    public Search() {
    }
    public void add(Position pos){
        searchList.add(pos);
    }

    public int size(){
        return searchList.size();
    }

    public Position get(int i){
        return searchList.get(i);
    }

    public void ParallelSearch(double lat, double lon, double ma, double width, double
length, double ts){
    /** Generate Parallel Search Pattern Positions from parameters **/
        int legs = (int)((length-ts)/ts)/2;
        double legLength = width - ts;
        double dist[] = new double[4];
        double tk[] = new double[4];
        double curLat = lat;
        double curLon = lon;
        int posCount = 0;
        int i, j;
        Position newPos = new Position();

        dist[0] = legLength;
        dist[1] = ts;
        dist[2] = legLength;
        dist[3] = ts;

        tk[0] = ma - 90;
        if (tk[0] < 0){tk[0] = tk[0] + 360;};
        tk[1] = ma;
        tk[2] = tk[0] + 180;
        if (tk[2] > 360){tk[3] = tk[3] - 360;};
        tk[3] = ma;


        for (i = 0;i < legs; i++){
            for(j=0;j<4;j++){
                newPos = NavClass.posFromDistBrg(curLat, curLon, dist[j], tk[j]);
                curLat = newPos.getLat();
                curLon = newPos.getLon();
                searchList.add(newPos);
            }
        }
        newPos = NavClass.posFromDistBrg(curLat, curLon, dist[0], tk[0]);
        searchList.add(newPos);
    }

    public void SectorSearch(double lat, double lon, double theta, double radius, double
crs){
        double cll = (radius / 60.0) * theta; // Cross Leg Length
        double ncl = 180.0 / theta;           // # Number of Cross Legs
        double nlegs = ncl * 2;               // # Number of Legs
        double cca = (theta / 2.0) + 90.0;    // # Course Change Angle
        int cllCount = 0;                     // # Keep track of cross legs
        double tk = crs;
        double dist[] = new double[4];
        Position newPos = new Position();
        int x = 0;

        double curLat = lat;
        double curLon = lon;

        dist[0] = radius;
        dist[1] = cll;
        dist[2] = radius;
        while (cllCount < ncl){
            for (x=0; x < 4; x++){
                newPos = NavClass.posFromDistBrg(curLat, curLon, dist[x], tk);
                curLat = newPos.getLat();
                curLon = newPos.getLon();
                searchList.add(newPos);

                if (x < 2){
                    tk += cca;
                    if (tk >= 360){
                        tk = tk - 360.0;
                    }
                }
            }

            cllCount += 1;
        }
    }

    public void ExpSquareSearch(double lat, double lon, double sqCount, double crs,
double ts){
    /** Generate Parallel Search Pattern Positions from parameters **/
        double legLength = ts;
        double tk = crs;
        double curLat = lat;
        double curLon = lon;
        Position newPos = new Position();

        int i, j;


        for (i = 0;i < sqCount; i++){
            for(j=0;j<4;j++){
                newPos = NavClass.posFromDistBrg(curLat, curLon, legLength, tk);
                curLat = newPos.getLat();
                curLon = newPos.getLon();
                searchList.add(newPos);

                tk+= 90;
                if (tk >= 360){
                    tk = tk - 360;
                }

                if (j==1){
                    legLength+= ts;
                }
                if (j==3){
                    legLength+= ts;
                }
            }
        }

    }


}
```

D. NAV CLASS

```
/*
 * NavClass.java
 *
 * Created on September 7, 2006, 9:52 AM
 *
 */
```

```
package mil.uscg.nav;
import java.util.ArrayList;

/**
 *
 * @author Bob Creigh creigh
 */
public class NavClass {

    /** Creates a new instance of NavClass */
    public NavClass() {
    }

    public static Position posFromDistBrg(double aLat, double aLon, double aDist, double
aBrg) {
    /** Calculate the Lat and Lon with a Distance and Bearing **/
        Position newPos = new Position(0,0);
        double lat = Math.toRadians(aLat);
        double lon = 0-Math.toRadians(aLon);
        double dist = (Math.PI/(180*60))*aDist;
        double brg = Math.toRadians(aBrg);

        double newLat;
        double newLon;

        newLat =
Math.toDegrees(Math.asin(Math.sin(lat)*Math.cos(dist)+Math.cos(lat)*Math.sin(dist)*Math.c
os(brg)));

       newLon = 0-Math.toDegrees(((lon - Math.asin(Math.sin(brg) *
Math.sin(dist)/Math.cos(lat))+Math.PI) % (2*Math.PI))-Math.PI);

        newPos.setLat(newLat);
        newPos.setLon(newLon);

        return newPos;
    }

}
```

E. POSITION CLASS

```
/*
 * Position.java
 *
 * Created on September 7, 2006, 10:17 AM
 *
 */

package mil.uscg.nav;

/**
 *
 * @author Bob Creigh
 */
public class Position {
    private double Lat;
    private double Lon;

    /** Creates a new instance of Position */
    public Position(double aLat, double aLon) {
        Lat = aLat;
        Lon = aLon;
    }

    public Position() {
    }

    public void setLat(double aLat){
        Lat = aLat;
    }

    public void setLon(double aLon){
        Lon = aLon;
    }

    public double getLat(){
        return Lat;
    }

    public double getLon(){
        return Lon;
    }

    public static String LatToDegMin(double lat){
        double deg = Math.floor(lat);
        double min = (lat - deg) * 60;
        String out = "";

        if (deg > 0){
            out = String.format("N%02d - %#04f",(int)deg, min);
        }
        else{
            out = "S"+ (int)deg + "-" + min;
        }

        return out;
    }

    public static String LonToDegMin(double lon){
        double deg = Math.floor(lon);
        double min = (lon - deg) * 60;
        String out = "";

        if (deg > 0){
            out = "E"+ (int)deg + "-" + min;
        }
        else{
            out = "W"+ (int)deg + "-" + min;
        }

        return out;
    }
}
```

F. WEB SERVICE DESCRIPTION LANGUAGE (WSDL)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://nav.uscg.mil/" name="SearchPatternService"
xmlns:tns="http://nav.uscg.mil/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://nav.uscg.mil/"
schemaLocation="SearchPatternService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="parallelSearchWS">
    <part name="parameters" element="tns:parallelSearchWS"/>
  </message>
  <message name="parallelSearchWSResponse">
    <part name="parameters" element="tns:parallelSearchWSResponse"/>
  </message>
  <message name="sectorSearchWS">
    <part name="parameters" element="tns:sectorSearchWS"/>
  </message>
  <message name="sectorSearchWSResponse">
```

```
    <part name="parameters" element="tns:sectorSearchWSResponse"/>
  </message>
  <message name="squareSearchWS">
    <part name="parameters" element="tns:squareSearchWS"/>
  </message>
  <message name="squareSearchWSResponse">
    <part name="parameters" element="tns:squareSearchWSResponse"/>
  </message>
  <portType name="SearchPattern">
    <operation name="parallelSearchWS">
      <input message="tns:parallelSearchWS"/>
      <output message="tns:parallelSearchWSResponse"/>
    </operation>
    <operation name="sectorSearchWS">
      <input message="tns:sectorSearchWS"/>
      <output message="tns:sectorSearchWSResponse"/>
    </operation>
    <operation name="squareSearchWS">
      <input message="tns:squareSearchWS"/>
      <output message="tns:squareSearchWSResponse"/>
    </operation>
  </portType>
  <binding name="SearchPatternPortBinding" type="tns:SearchPattern">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="parallelSearchWS">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="sectorSearchWS">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="squareSearchWS">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="SearchPatternService">
    <port name="SearchPatternPort" binding="tns:SearchPatternPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. SEARCH PATTERN CLIENT SOURCE CODE

A. CODE OVERVIEW

This code provides a GUI client that can be used by any OS that supports the Java Virtual Machine. Swing is a GUI toolkit for Java. It is one part of the Java Foundation Classes. Swing includes GUI widgets such as text boxes, buttons, split-panes, and tables. This client is one way to consume the Search Pattern Web service.

B. SEARCH PATTERN SWING CLIENT

```
/*
 * SearchForm.java
 *
 * Created on November 27, 2006, 8:16 AM
 */

package mil.uscg.searchclient;

/**
 *
 * @author  bob
 */
public class SearchForm extends javax.swing.JFrame {

    /** Creates new form SearchForm */
    public SearchForm() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">//GEN-
BEGIN:initComponents
    private void initComponents() {
        btnGrpSearchType = new javax.swing.ButtonGroup();
        jPanel1 = new javax.swing.JPanel();
        jRadParallel = new javax.swing.JRadioButton();
        jRadSector = new javax.swing.JRadioButton();
        jRadExpSq = new javax.swing.JRadioButton();
        jPanel2 = new javax.swing.JPanel();
        jLblLat = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLblLen = new javax.swing.JLabel();
        jLblWidth = new javax.swing.JLabel();
        jLblTs = new javax.swing.JLabel();
        jLblMa = new javax.swing.JLabel();
        jTxtLat = new javax.swing.JTextField();
        jTxtLon = new javax.swing.JTextField();
        jTxtLen = new javax.swing.JTextField();
        jTxtWidth = new javax.swing.JTextField();
        jTxtTs = new javax.swing.JTextField();
        jTxtMa = new javax.swing.JTextField();
        jPanel3 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTxtAreaResults = new javax.swing.JTextArea();
        jToggleButton1 = new javax.swing.JToggleButton();
        jToggleButton2 = new javax.swing.JToggleButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Search Type"));
        btnGrpSearchType.add(jRadParallel);
        jRadParallel.setSelected(true);
        jRadParallel.setText("Parallel");
        jRadParallel.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
        jRadParallel.setMargin(new java.awt.Insets(0, 0, 0, 0));
        jRadParallel.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jRadParallelActionPerformed(evt);
            }
        });

        btnGrpSearchType.add(jRadSector);
        jRadSector.setText("Sector");
        jRadSector.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
        jRadSector.setMargin(new java.awt.Insets(0, 0, 0, 0));
        jRadSector.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jRadSectorActionPerformed(evt);
            }
        });

        btnGrpSearchType.add(jRadExpSq);
        jRadExpSq.setText("Expanding Square");
        jRadExpSq.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
        jRadExpSq.setMargin(new java.awt.Insets(0, 0, 0, 0));
        jRadExpSq.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jRadExpSqActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout jPanel1Layout = new
org.jdesktop.layout.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(org.jdesktop.layout.GroupLayout.TRAILING,
jPanel1Layout.createSequentialGroup()
                .add(34, 34, 34)
                .add(jRadParallel)
                .add(66, 66, 66)
                .add(jRadSector)
                .add(50, 50, 50)
                .add(jRadExpSq)
                .addContainerGap(35, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel1Layout.createSequentialGroup()

.add(jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jRadExpSq)
                    .add(jRadSector)
                    .add(jRadParallel))
                .addContainerGap(8, Short.MAX_VALUE))
        );

        jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder("Search
Parameters"));
```

```
        jLblLat.setText("Latitude");

        jLabel2.setText("Longitude");

        jLblLen.setText("Length");

        jLblWidth.setText("Width");

        jLblTs.setText("Track Space");

        jLblMa.setText("Major Axis");

        org.jdesktop.layout.GroupLayout jPanel2Layout = new
org.jdesktop.layout.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(org.jdesktop.layout.GroupLayout.TRAILING,
jPanel2Layout.createSequentialGroup()

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jLblTs)
                    .add(jLblLat)
                    .add(jLblLen))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(jTxtTs, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 93,
Short.MAX_VALUE)
                    .add(jPanel2Layout.createSequentialGroup()
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(jTxtLat, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 93,
Short.MAX_VALUE))
                    .add(jTxtLen, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 93,
Short.MAX_VALUE))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED, 68,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING, false)
                    .add(org.jdesktop.layout.GroupLayout.LEADING,
jPanel2Layout.createSequentialGroup()
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(jLabel2, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 78,
Short.MAX_VALUE))
                    .add(org.jdesktop.layout.GroupLayout.LEADING, jLblWidth,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 78, Short.MAX_VALUE)
                    .add(org.jdesktop.layout.GroupLayout.LEADING, jLblMa,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING, false)
                    .add(jTxtLon, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 93,
Short.MAX_VALUE)
                    .add(jTxtWidth)
                    .add(jTxtMa))
                .addContainerGap())
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel2Layout.createSequentialGroup()

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jLblLat)
                    .add(jTxtLat, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
```

```
                    .add(jTxtLon, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(jLabel2))
                .add(21, 21, 21)

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jLblLen)
                    .add(jTxtLen, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(jTxtWidth, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(jLblWidth))
                .add(26, 26, 26)

.add(jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jLblTs)
                    .add(jTxtTs, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(jTxtMa, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(jLblMa))
                .addContainerGap(12, Short.MAX_VALUE))
        );

        jPanel3.setBorder(javax.swing.BorderFactory.createTitledBorder("Results"));
        jTxtAreaResults.setColumns(20);
        jTxtAreaResults.setRows(5);
        jScrollPane1.setViewportView(jTxtAreaResults);

        org.jdesktop.layout.GroupLayout jPanel3Layout = new
org.jdesktop.layout.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(
            jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel3Layout.createSequentialGroup()
                .addContainerGap()
                .add(jScrollPane1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 396,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(31, Short.MAX_VALUE))
        );
        jPanel3Layout.setVerticalGroup(
            jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jPanel3Layout.createSequentialGroup()
                .add(jScrollPane1, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 227,
Short.MAX_VALUE)
                .addContainerGap())
        );

        jToggleButton1.setText("Generate");
        jToggleButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jToggleButton1ActionPerformed(evt);
            }
        });

        jToggleButton2.setText("Clear");
        jToggleButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jToggleButton2ActionPerformed(evt);
            }
        });

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(getContentPane());
```

```
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
layout.createSequentialGroup()
                        .add(jToggleButton2)
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(jToggleButton1))

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING, false)
                        .add(org.jdesktop.layout.GroupLayout.LEADING, jPanel3, 0,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .add(org.jdesktop.layout.GroupLayout.LEADING, jPanel1,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .add(org.jdesktop.layout.GroupLayout.LEADING, jPanel2,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                .addContainerGap(20, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(jPanel1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 54,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jPanel2, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 153,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jPanel3, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED, 13,
Short.MAX_VALUE)
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(jToggleButton1)
                    .add(jToggleButton2))
                .addContainerGap())
        );
        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jToggleButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jToggleButton2ActionPerformed
        // Clear Fields
        jTxtLat.setText("");
        jTxtLon.setText("");
        jTxtWidth.setText("");
        jTxtLen.setText("");
        jTxtMa.setText("");
        jTxtTs.setText("");
        jTxtAreaResults.setText("");
    }//GEN-LAST:event_jToggleButton2ActionPerformed

    private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jToggleButton1ActionPerformed
        // Get Search Results from WS
        double lat = 0;
        double lon = 0;
        double ma = 0;
        double width = 0;
        double length = 0;
        double ts = 0;
        double radius = 0;
        double theta = 0;
        double crs = 0;
        double sqCount = 0;
        int i = 0;

        lat = Double.valueOf(jTxtLat.getText());
        lon = Double.valueOf(jTxtLon.getText());

        if (jRadParallel.isSelected()){

            ma = Double.valueOf(jTxtMa.getText());
            width = Double.valueOf(jTxtWidth.getText());
            length = Double.valueOf(jTxtLen.getText());
            ts = Double.valueOf(jTxtTs.getText());

            try { // Call Web Service Operation
                me.searchclient.SearchPatternService service = new
me.searchclient.SearchPatternService();
                me.searchclient.SearchPattern port = service.getSearchPatternPort();

                // process result here
                java.lang.Object result = port.parallelSearchWS(lat, lon, length, width,
ma, ts);
                jTxtAreaResults.setText(result.toString());
            } catch (Exception ex) {
                // display exceptions here
                jTxtAreaResults.setText(ex.toString());
            }

        }

        if (jRadSector.isSelected()){

            theta = Double.valueOf(jTxtWidth.getText());
            radius = Double.valueOf(jTxtLen.getText());
            crs = Double.valueOf(jTxtTs.getText());
            try { // Call Web Service Operation
                me.searchclient.SearchPatternService service = new
me.searchclient.SearchPatternService();
                me.searchclient.SearchPattern port = service.getSearchPatternPort();

                // process result here
                java.lang.Object result = port.sectorSearchWS(lat, lon, theta, radius,
crs);
                jTxtAreaResults.setText(result.toString());
            } catch (Exception ex) {
                // display exceptions here
                jTxtAreaResults.setText(ex.toString());
            }

        }

        if (jRadExpSq.isSelected()){

            crs = Double.valueOf(jTxtWidth.getText());
            sqCount = Double.valueOf(jTxtLen.getText());
            ts = Double.valueOf(jTxtTs.getText());
            try { // Call Web Service Operation
                me.searchclient.SearchPatternService service = new
me.searchclient.SearchPatternService();
                me.searchclient.SearchPattern port = service.getSearchPatternPort();

                // process result here
                java.lang.Object result = port.squareSearchWS(lat, lon, sqCount, ts,
crs);
                jTxtAreaResults.setText(result.toString());
            } catch (Exception ex) {
                // display exceptions here
                jTxtAreaResults.setText(ex.toString());
            }
        }
    }//GEN-LAST:event_jToggleButton1ActionPerformed

    private void jRadParallelActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jRadParallelActionPerformed
        // Setup fields for Parallel Search
        jLblLen.setText("Length");
        jLblWidth.setText("Width");
        jLblTs.setText("Track Space");
        jTxtMa.setVisible(true);
        jLblMa.setVisible(true);
        jLblTs.setVisible(true);
        jTxtTs.setVisible(true);
    }//GEN-LAST:event_jRadParallelActionPerformed

    private void jRadExpSqActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jRadExpSqActionPerformed
        // Setup fields for Expanding Square Search
        jLblLen.setText("Cycles");
        jLblWidth.setText("Initial Track");
        jLblTs.setText("Track Space");
        jTxtTs.setVisible(true);
        jTxtMa.setVisible(false);
        jLblMa.setVisible(false);
    }//GEN-LAST:event_jRadExpSqActionPerformed

    private void jRadSectorActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jRadSectorActionPerformed
        // Setup fields for Sector Search
        jLblLen.setText("Radius");
        jLblWidth.setText("Theta");
        jLblTs.setText("Initial Track");
        jTxtMa.setVisible(false);
        jLblMa.setVisible(false);
    }//GEN-LAST:event_jRadSectorActionPerformed

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new SearchForm().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.ButtonGroup btnGrpSearchType;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLblLat;
    private javax.swing.JLabel jLblLen;
    private javax.swing.JLabel jLblMa;
    private javax.swing.JLabel jLblTs;
    private javax.swing.JLabel jLblWidth;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JRadioButton jRadExpSq;
    private javax.swing.JRadioButton jRadParallel;
    private javax.swing.JRadioButton jRadSector;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JToggleButton jToggleButton1;
    private javax.swing.JToggleButton jToggleButton2;
    private javax.swing.JTextArea jTxtAreaResults;
    private javax.swing.JTextField jTxtLat;
    private javax.swing.JTextField jTxtLen;
    private javax.swing.JTextField jTxtLon;
    private javax.swing.JTextField jTxtMa;
    private javax.swing.JTextField jTxtTs;
    private javax.swing.JTextField jTxtWidth;
    // End of variables declaration//GEN-END:variables

}
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dan Boger
   Naval Postgraduate School
   Monterey, California

4. Rick Hayes-Roth
   Naval Postgraduate School
   Monterey, California

5. Rex Buddenberg
   Naval Postgraduate School
   Monterey, California

6. Russell Dash
   Naval Postgraduate School
   Monterey, California

7. Robert Creigh
   Naval Postgraduate School
   Monterey, California

8. Chris Gunderson
   Naval Postgraduate School
   Monterey, California

9. Dave Reading
   Naval Research Laboratory
   Washington, D.C.

10. CDR Marc Sanders
    Coast Guard Headquarters
    Washington, D.C.